The Software Metrician's Workbench

A System for Supporting Software Metrics Research.

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Master of Science in Computer Science

by

S. R. Garner

University of Canterbury

1991

Table of Contents.

Abstract.

The desire to produce software of better quality has lead to the requirement for better management of the software development process. Software development has been likened to an engineering discipline, in that it uses similar analysis techniques to produce a product. However, unlike other engineering fields the software development field is lacking in quantitative measures and models for describing characteristics and effects in development and the product produced.

Software metrics is a relatively new field that is attempting to develop measures and models in which to use those measurements, in order to improve the management of software development. Unfortunately, the current state of metrics suffers from a lack of validation and testing of measures proposed, and a lack of historical data on which base development of new metrics.

The Software Metrician's Workbench is an environment that has been developed to aid in the development of new metrics and the evaluation of existing ones. It is based upon a database management system, that is used to store historical program and metric data, and is designed to be extended by the user through the addition of new analysis and data collection tools that use a common data interface.

In this thesis, a discussion of the current state of software metrics is given, including descriptions of other automated metrics tools. The aims behind the design and implementation of the Software Metrician's Workbench system are described, and examples of the system's use are given. The future of the Software Metrician's Workbench is then discussed, including it's current use in metric research, and proposed enhancements.

Chapter 1. Quality Software and Software Metrics.

1.1 Quality Software.

As the price-performance ratio for computer hardware continues to rapidly improve the proportion of the cost of a computer system that is made up by software is increasing. The actual cost in of the software in a computer system has been estimated as being as high as 80% of the cost of the overall system. Of that cost the bulk of it (60%) is spent on maintaining the installed software[1]. This development has lead to what commentators call the "Software Crisis". This situation is characterized by:

- Highly inaccurate schedules and cost estimates.
- Software of poor quality.
- A productivity rate that is increasing more slowly than the demand for software.

Obviously if the quality of the software is increased then the proportion of the cost due to maintenance should go down. With less maintenance more resources become available for new software development. Part of the reason for the "crisis" situation is the developers lack in ability to isolate those factors present in the software development process and products that causes a lack of quality. Isolation at maintenance stage is possible, but with up to 60-70% of all implementation errors being due to design faults[2] early prediction of characteristics that define quality, or the lack of it, in software is highly desirable.

Quality software can be defined as software which displays the following qualities[3]:

• Functionality	The software achieves what it was specified to do.
• Reliability	The software can be depended on to work within the conditions and environment specified.
• Performance	The software functions correctly and performs it's functions at a useful speed.

---

[1]Conte, S D, Dunsmore, I and Shen H E; Software Engineering Metrics and Models; *Benjamin Cummings, Menlo Park;* 1986

[2]Weiss, D M and Basili, V R; Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory; *EEE Trans. Software Engineering;* Vol. 11; 157-168; 1985

[3]Mills, E E; Software Metrics: SEI Curriculum Module SEI-CM-12-1.1; *Carnegie Mellon University;* 1988

- Economy — Does the software make the best use of the resources available and is it the best software that can be obtained with the resources available for development?
- Maintainability — Can the software be easily understood and modified?

All of these characteristics will be present in different quantities in all software but different applications require different mixes. For example, real-time processing might require such high performance that the product has to compromise on coding the software in such a way as to make it more understandable.

The main aim should be to identify what quality characteristics are important for the project and then to make sure that the software contains those characteristics. Prediction early in the development process can help to guide decisions and isolate those areas where those qualities are missing or will cause a lack of quality later.

1.2 Development of Software.

Software development is usually viewed not as a single process but rather as an overlapping sequence of processes. Typically these are grouped into requirements analysis and specification, design, implementation or coding, system testing, installation and maintenance. Ideally each of these sub-processes or phases in the development process are carried on one after the other, the next one commencing upon the completion of the previous process. For example, coding not started until the design of the system is completely finished. Each phase or process may be sub-divided into component processes as well, such as dividing the design process into system design and detailed design. System design is where the overall structure of the program is defined, in terms what components such as modules, files, databases will be needed, and detailed design where the behavior of each design component is specified using a design notation such as pseudocode or action charts.

Practically though, several or all of the processes can be going on the some extent simultaneously. As well as earlier processes affecting the later processes, a process can return feedback to earlier process as well. An example of this is the discovery during the implementation phase of development of a error in the specification or design of the software. This can cause the erroneous area of the software to be re-designed which then will cause re-coding.

It can be seen therefore that development is not a single linear sequence of steps but rather a cycle, with processes having effects on earlier processes as well as later ones (Figure 1.1). The maintenance phase of software development can almost be treated as a complete copy of this software development life cycle in it's own right. When extra functionality is to be

added to the implemented product or an error fixed then it may be necessary to re-specify the requirements, re-design portions of the software, code the changes, test the changes, and then maintain those changes.

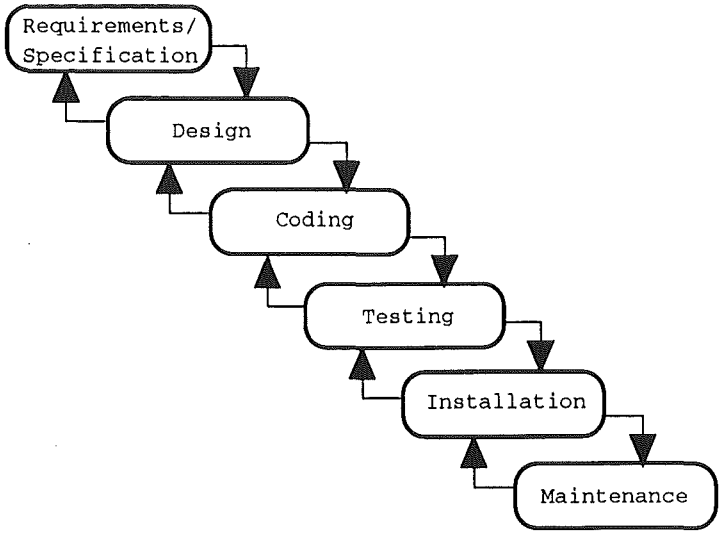


Figure 1.1 Software Development Life Cycle.

1.3 Software Engineering

This development of software through the use of a sequence of well-defined production processes has much in common with other production engineering disciplines. Indeed, the term "software engineering" has come into use to describe the variety of tools and techniques that allow the production of cost-effective, reliable software within specified time constraints.[4] However, one of the characteristics of other engineering disciplines is that of being able to quantitatively measure relevant processes and products, and in this respect software development is sorely lacking. For example, measurements can be made on scale models of a bridge to determine it's maximum load capacity under different conditions, or the efficiency of two chemical reactions that produce the same product can be compared.

The reasons that these things are able to be done in other disciplines is through the use of well-define measurements of product and process characteristics, and the development of models to interpret those measurements. These models are based upon experimentation and the use of bodies of historical data containing observations many similar products and processes.

Software development, being a relatively new activity which is undergoing rapid change as new methodologies and tools are developed, has little historical data with which to base models on and that information that is available tends to be obtained from subjective or

[4]Conte, S D, Dunsmore, I and Shen H E; Software Engineering Metrics and Models

ill-defined measurements with no corresponding explanatory models. Without the measures and their corresponding models, then the ability to describe what is happening within software process or product, and to use that information to predict and control the development process, is severely limited.

Software metrics is a relatively new field, that attempts to bring quantitative measurement and analysis to software development.

1.4 Software Metrics.

Software metrics is the field of research concerned with developing quantitative measures and models in describing the software development process, and characteristics of development product. These measures and models can then be used to predict the quality of products earlier in the life cycle, identify maintenance problems, evaluate the effects of different development methodologies, and provide the developer with estimates of resource allocation required for product development.

A software metric, or just metric, is a particular measurement of a characteristic of a product or process. The software metric, when applied, will produce a value (or set of values) that describes of the characteristic being measured. More often than not these metric values are numerical.

Software metrics can be divided into two main families - product and process metrics.

Product metrics are measures of the software product at any stage of it's development, from requirements to installed system[5]. The software product is the result of the processing done during a specific phase of software development. For example, the software product generated by the design phase might be documents that describe the system using notations such as Entity-Relationship diagrams, Data Flow diagrams and Structure charts. In the coding or implementation phase the software product include source code, object code and documentation. Product metrics measure characteristics of these products and then produce values that can be interpreted by the appropriate model. Examples of product metrics are ones that measure the complexity of a design, or the size in statements of the source code of the final product.

The other family of metrics are process metrics. These are measures of the software development process, such as the overall development time, type of methodology used, or average level of experience of the programming staff[6]. Process metrics typically include time as a component of the measure, such as calculating the number of lines of code

[5]Mills, E E; Software Metrics: SEI Curriculum Module SEI-CM-12-1.1

[6]Mills, E E; Software Metrics: SEI Curriculum Module SEI-CM-12-1.1

produced per month per programmer, and may include product metrics as components as well. Process metrics are used to estimate resource usage for a project such as effort required or monetary cost, as well as productivity of a developer or group of developers.

In order for a particular metric to be useful various criteria have to be met. The criteria vary between the promoters of metrics but a basic subset of criteria is[7]:

- Metric should be simple and precisely defined.
- Metric should be objective.
- Metric should be easily obtainable.
- Metric should be valid.
- Metric should be robust.

These criteria lead to a metric that's evaluation clearly understood, whose value doesn't change when calculated by a different observer or at a different time, is not so complex that it takes too much money or time to calculate, measures what it intends to measure, and is relatively insensitive to insignificant changes in the process or product. Other definitions of criteria include adding a dialogue between computer scientists, software developers and cognitive scientists to get a better model of the software development process and of complexity in software.

Ideally, the level of subjectivity can be reduced by automating the metric with some algorithmic procedure, which removes the observer from the measurement process.

The main problems that software metrics suffer from are:

- Lack of confidence between software researchers and developers.
- Misuse of metrics.
- Lack of empirical evidence for use of a metric.
- Lack of good models to interpret the metric results.

In the first case the has been brought about by the lack of validation of metrics, and a lack of dialogue between those who develop metrics and those who have to use them. Some widely used metrics, such as Halstead's Software Science, have been shown to be super-sensitive to the items measured, resulting in frustration for users of the metrics. Also there is an

---

[7]Mills, E E; Software Metrics: SEI Curriculum Module SEI-CM-12-1.1

inherent distrust between the academic metrics researchers and the commercial software developers. This has not been helped by the use of metrics to measure programmer productivity, which if the metric being used does not take factors such as complexity of the task involved into account, can lead to distorted figures. Also there has been the occasion for promoters of metrics packages to promote these packages as a panacea for the management of development without educating the users in how to interpret the results, leading to a distrust in the field.

A very real problem with metrics research is obtaining empirical data used for validating metrics. Small test programs behave much differently to multi-programmer, multi-module software system, making results obtained from validating small sized programs open to criticism, and more sensitive to small changes in the programs. Getting large sets of data is a problem as much of it is commercially sensitive, though in Chapter 3 a system is discussed that the authors obtained large quantities of real world data for[8]. Also some validation of method metrics would require the parallel development of several software systems using different methodologies, obviously costing too much.

The last problem is that once large quantities of metric data have been extracted then the model used to interpret the results may be incorrect. Thus better models need to be developed and this won't be achieved without a large body of historical metric data to experiment with. Also if a large test set of program data is available fro experimentation that a common benchmark set of values may be determined, and new metrics that are developed can be compared with other measurements made with different metrics for the same programs in the same development environment.

The Software Metrician's Workbench (SMW) system is a project that is aimed at helping to remedy some of the problems associated with the lack of empirical data, program data, and benchmarks.

1.5 SMW - The Software Metrics Workbench.

The SMW system is a flexible, extensible and powerful workbench for software metricians (software metrics researchers). The SMW system is designed to support research into software product metrics though the use of a relational database system being used to maintain a historical archive of programs and product metric information. As well it supports automated metric collection tools allowing a consistent, objective metric data to be collected and stored for many programs. It aims to provide facilities to it's users that allow the following activities:

---

[8]Yu, T J, Nejmeh, H E, Dunsmore, H E and Shen, V Y; SMDC: An Interactive Software Metrics Data Collection and Analysis System; *J. Systems and Software*; Vol. 8; 39-46; 1988

1.5.1 Storage of program data.

The SMW system is designed to be able to extract characteristics of software through the use of different tools and to store the characteristics. This program information will be available for be used in the calculation of metrics, as well as providing information on the structure of the software. Storage of the program data separate from the software product minimizes the disruption to development that would occur when the product was accessed for metric value calculation. Multiple versions of the same program can be stored within the system to allow tracking of software as it is developed, and multiple different programs can also be stored together.

1.5.2 Storage of metric data.

Metric values calculated for products stored in the SMW system may also be stored within the SMW system for retrieval at a later date for reporting purposes of for inclusion in other metric calculations.

1.5.3 A consistent data interface.

A data interface that allows the users to access the data through standard protocols, maintaining the integrity of the data, and allowing the development of program data collection, metric analysis, and data retrieval tools. By using the data interface protocols users can develop new tools without having to be concerned with the physical structure of how the data is stored, but rather it's logical or conceptual structure.

1.5.4 Basic tool set

A basic set of tools that provide facilities for data capture, both program and metric data, for administration of the SMW system, for data retrieval from the system using a query language, and a prototype user interface for examining the contents of the SMW system. The latter tool provides an example of how data may be retrieved from the SMW system and then transferred into other tools, such as a graphing or statistic package, either directly from inside the user interface, or via the reporting mechanism.

1.5.5 Use of the SMW system.

The SMW system is envisaged as a repository of metric and program data that will increase in size as metrics research with it continues. This will allow a set of empirical data to be available for use in validating metrics, as well as providing data for comparisons between metrics calculated for different programs and for different metrics on the same program. New metrics collection tools can be added and old ones updated without the program data having to be updated or product re-analyzed, and the query language support allows flexible

access to the metric data. Figure 1.2 shows the part of the interactive user interface used for displaying call relationships between modules in a program.

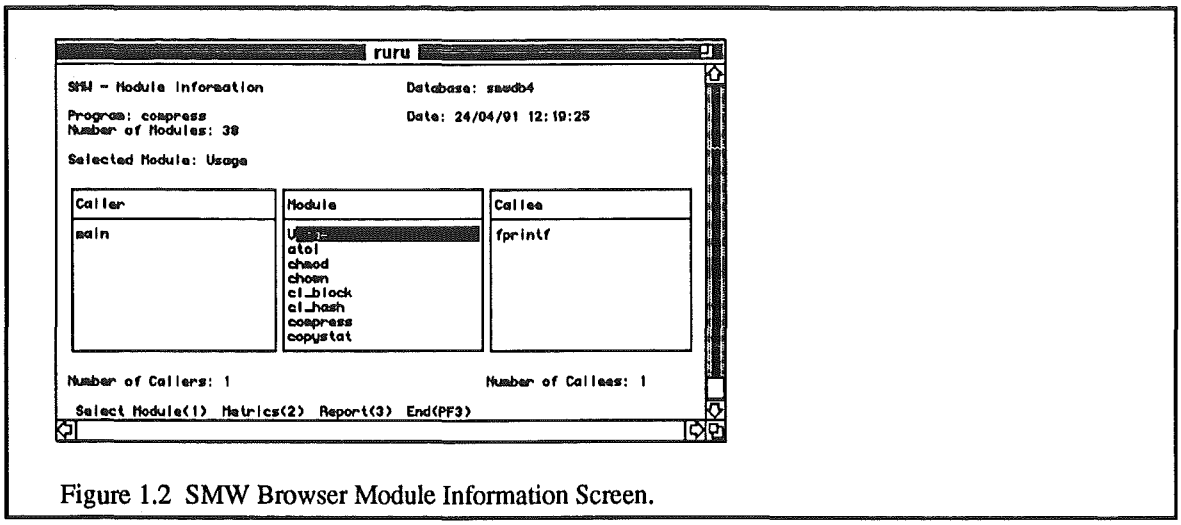


Figure 1.2 SMW Browser Module Information Screen.

1.6 Structure of this thesis.

The chapters following contain discussions of the following subjects. Firstly there is a discussion of software product metrics, including their applications and deficiencies, followed by a chapter on a variety of automated metrics systems and methodologies. Then the SMW design and implementation is discussed in the next three chapters, and that is followed by a chapter detailing studies carried out with SMW. Finally there is a summary of SMW, it's future direction, and a conclusion about the SMW system.

Chapter 2 History of and Trends in the Software Product Metrics Field.

2.1 Introduction

This chapter gives an overview of software product metrics. It covers several important early metrics, and discusses metrics from the view point of what stage in the software development life cycle they can be applied at. The latter part of the chapter deals with trends in the last fifteen or so years that metrics research has been conducted with, and the new directions that metrics are taking.

Software metrics as a research field in it's own right can into existence in the last part of the 1970's with the establishment of three metrics that have become know as the "Classic" product metrics. This is not to say that software product measurement was not being performed earlier that this, but rather the earlier measurements were normally incidental to the main research effort, rather than the aim of the research. An example of this is an experiment conducted by Knuth[9] which consisted of measuring FORTRAN programs using token and statement counts in order to describe how FORTRAN programmers programmed so that compiler design could be improved.

The next section describes the "Classic" metrics and discusses their drawbacks, and is then followed by discussions of metric families that are available at the specification, system design and implementation phases of software development.

2.2 The "Classic" Product Metrics.

The "Classic" metrics are lines of code, Halstead's Software Science, and McCabe's cyclomatic complexity. They were the main three software metrics to be developed and used for the measurement of program complexity in the mid to later 1970's and early 1980's. Their ease of automation has lead them to be used relatively widespreadly in spite of deficiencies in the models upon which they were built. Indeed, they are readily and cheaply available as part of commercial products[10,11], or from various public domain sources for a variety of hardware platforms and operating systems.

2.2.1 Lines of Code

The lines of code metric, LOC (KLOC - thousands of lines of code), is probably the most

---

[9]Knuth, D E; An Empirical Study of FORTRAN Programs; *Software- Practice and Experience;* Vol. 1; 105-135; 1971.

[10]PC-Metric; SET Laboratories Inc., P.O. Box 868, Mulino, OR 97042, U.S.A.

[11]Code-Check; Abraxas Software Inc., 7033 SW Macadam Ave., Portland, OR 97219, U.S.A.

widespread metric. It is used as both a product metric for measuring the size of programs and program units, and as a process metric, normally a similar form to LOC per month per programmer. In measuring program size it has been used as a predictor for both reliability and ease of maintenance.

The main difficulty in measuring lines of code is not the actual data collection but rather the definition of what to measure. For example, are the following included in the total?

- comments.
- blank lines.
- multi-statement lines (are these treated as only one line?).
- number of executable source statements only.
- lines containing declarative statements.
- number of machine code instructions generated.

The LOC measure has been defined by one author as:

"A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on a line. This specifically includes all lines containing program headers, declarations, and executable and non-executable statements."[12]

The main drawbacks of the metric are that it doesn't take into account the complexity of each line counted and it is very susceptible to the formatting of the product being measured, especially in free-form languages like C where statements can be spread over many lines, or an entire program may exist on only one line. Also for the product to be measured it has to have been coded and as such the metric is not available as a predictor of the quality of the product, rather a measure of what already exists.

On the benefit side LOC is easy to calculate, can in some circumstances perform better that other metrics as a predictor of error-proneness, and is able to be used as a baseline measure to compare other metrics against[13]. Pseudocoded modules may be able to give an indication of the size in LOC provided it's size correlates well with the actual final size of the module. Chapter 7 contains results of a study on correlation between several metrics and a number of

---

[12]Conte, S D, Dunsmore, I and Shen H E; Software Engineering Metrics and Models

[13]Basili, V R and Hutchens, D H; An empirical Study of a Syntactic Complexity Family; *IEEE Trans. Software Engineering;* Vol SE-9; No. 6; 664-672; 1983

statements metric.

2.2.2 Halstead's Software Science.

In 1977 Maurice Halstead published the Software Science metric[14]. This was a set of metrics that fitted into an overlying framework for measuring and explaining software complexity. It was the first such framework to be developed, and even though it has been largely rejected now, it is still one of the most common set of metrics in use.

Software Science attempted to provide a coherent framework within which software can be measured. It attempts to do this by modelling the comprehension of a software product as the mental manipulation of program tokens. Each program can be seen as a continuous sequence of program tokens, where a token is either an operator, such as an executable program verb, or an operand, such as a variable or constant.

By using basic measurements of the number of program tokens, various computed measurements could be calculated such as difficulty of comprehension of the program.

The basic measurements used by software science are:

n_1 = count of unique operators

n_2 = count of unique operands

N_1 = total number of operators

N_2 = total number of operands

From these the program vocabulary, n, and the program length, N, can be calculated.

$n = n_1 + n_2$

$N = N_1 + N_2$

Software Science states that to understand the program the programmer or reader needs to manipulate the program tokens. Each token that needed to be manipulated has to be retrieved from a mental dictionary, and this dictionary would contain the entire program vocabulary. Access to the dictionary would be through using a mental binary search.

From this model the following computed metrics were be developed.

The number of comparisons or dictionary accesses required to understand a piece of

---

[14]Halstead, M H; Elements of Software Science; *Elsevier North-Holland, New York*; 1977

software is though of as the program volume, V, where

$$V = N * \log_2 n$$

The program level, L, is a measure of the program's volume against some optimal minimum volume, V^*. L was defined as

$$L = V^*/V \quad \text{where } V^* \text{ is the optimal minimum value for V.}$$

Because the it seemed natural to have programs with larger volumes having a higher level difficulty measure, D, was defined to give more complex programs a higher value than lower complexity ones.

$$D = 1/L$$

The optimal minimal program volume, V^*, is in fact impossible to determine so an estimator of L, $\hat{L}$ is used where

$$\hat{L} = 2/n_1 * n_2/N_2$$

which gives the estimator of D, $\hat{D}$, as

$$\hat{D} = n_1/2 * N_2/n_2$$

The difficulty is a measure of a program against the theoretical optimum program volume. The effort, E, to understand the program is

$$E = D * V$$

There are four main theoretical objections to this model of software complexity, and various evaluations and criticisms have been written about it[15,16].

Firstly, there are problems with the counting rules. The rules for counting tokens excluded input/output and declarative statements for no apparent reason. The definition of what is an operator and what is an operand is subjective and is left up to the individual measurer to decide. Also IF statements with compound conditions are treated in such a way cause the IF operator to count as a single operator, even though logically several comparisons occur.

---

[15]Lassez, J L, van der Knijff, D, Shepherd, J and Lassez, C; A Critical Examination of Software Science; *J. Systems and Software;* Vol. 2; 105-112; 1981

[16]Shen, V Y, Conte, S d and Dunsmore, H E; Software Science Revisited: A Critical Analysis of the Theory and its Empirical Support; *IEEE Trans. Software Engineering;* Vol SE-9; No. 2; 155-165; 1983

For example, if the following is changed from

```
IF (( A < B ) AND ( C < D )) THEN
```

to

```
IF ( A < B) THEN
      IF (C < D) THEN
```

then the number of operators has increase from one to two for exactly the same condition.

The second objection is to the psychological assumption that a binary search is used by the programmer within the context of programming, when there is no empirical evidence to support this.

The next objection is that the view of a program as a sequence of tokens is too simplistic and does not take into account the flow of control in the program, the physical structure of the program such as nesting, and the data structures involved.

The last objection is that the model was tested only on small programs. These programs were seven to fifty-nine statements in length and were only for small scale algorithms.

For these reasons it was decided that SMW would not include a Software Science metric analysis tool in it's basic tool set, but it's implementation for C source code using the scanner described in Chapter 6 could be easily done should the need arise for on later.

On the positive side Software Science was the first attempt to provide an integrated framework for the measurement of programs, and was easy to automate once the operators and operands were defined. Application of software science programs to software development have shown positive results in managing the development process sometimes, though whether this is due to the metrics being used or an increased awareness of the structure of the project because of applying the metric is unknown.

2.2.3 McCabe's Cyclomatic Complexity.

The cyclomatic complexity measure was published in 1976 by Thomas McCabe[17]. It differs from the previous two metrics in that it attempts to measure the complexity of the control flow graph within a program sub-unit. The metric was developed to aid in two areas. Firstly for the prediction of the effort required for a piece of software to be tested. And secondly to predict the complexity of related characteristics within a finished piece of

---

[17]McCabe, T J; A Complexity Measure; *IEEE Trans. Software Engineering;* Vol. SE-2; No. 4; 308-320; 1976

software.

The metric is based on the concept that software, or program sub-units can be viewed as a directed graph. Each edge represents the flow of control between two nodes, where nodes represent single statements or blocks of sequential statements. Given this method of viewing software it was hypothesized that program complexity could be related to the control flow complexity. That is the program units with more control structures within them, such as loops, jumps, and selections, would be more complex to understand.

The complexity of the control flow graph can be measured by calculating it's cyclomatic complexity. This is possible by treating the control flow graph as a strongly connected graph, where every node is reachable from every other node. In order to convert the control flow graph into a strongly connected graph the terminating node in the program sub-unit was connected to the starting or entry node of the sub-unit. A sample program unit is given in Figure 2.1 with it's associated control graph, with the extra edge added from the terminating node (END) to the entry node (BEGIN). This extra edge makes the calculation for cyclomatic complexity similar but not exactly the same as calculating the cyclomatic number for the graph.

The cyclomatic number, ν, of graph G is:

$$\nu(G) = e - n + 1$$

where e is the number of edges and n is the number of nodes.

For the cyclomatic complexity, ν, of graph G describing a program subunit's control flow graph the calculation is:

$$\nu(G) = e - n + 2p$$

where e is the number of edges not including the edge from the exit node to the entry node, n is the number of nodes and p is the number of connected components, such as procedures. For normal situations p is 1.

```
BEGIN
  REPEAT
    writeln('Enter a number or zero to stop');
    readln(num);
    IF num > 0 THEN
      writeln(num,'is positive')
    ELSE
      IF num < 0 THEN
        writeln(num,'is negative');
  UNTIL num = 0;
END
```

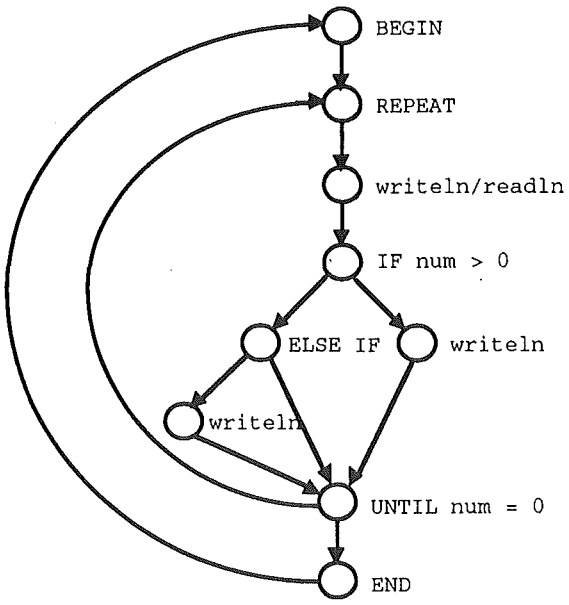


Figure 2.1 Program sub-unit with associated flow graph.

For example, the program unit in Figure 2.1 has the cyclomatic complexity of

$$v(G) = 11 - 9 + 2$$

$$= 4$$

For multi-module or program unit programs the v(G) for the whole program can be calculated as the sum of the v(G) values of each module[18].

McCabe observed that the calculation of cyclomatic complexity can be reduced to simply counting the number of decision statements and adding one to the total. For example, in Figure 2.1 the cyclomatic complexity can be calculated as

[18]Conte, S D, Dunsmore, I and Shen H E; Software Engineering Metrics and Models, 66-70

$$\nu(G) = 2\ IF + 1\ UNTIL + 1$$

$$= 4$$

In the case of multi-condition decision statements the decision statement counting rules can be extended to allow each condition to be treated as a separate decision statement. Multi-way decision statements, such as CASE statements add *n-1* to the cyclomatic complexity where *n* is the number of cases.

There are several notable drawbacks to the definition of this metric.

Firstly, for any linear sequence of non-decision statements $\nu(G)$ is always going to be one. Thus if a program unit consists of only two non-decision statements then that sequence will have the same $\nu(G)$ as a sequence of a thousand non-decision statements. Software that demonstrates this property tends to be "function-bound" rather than "decision-bound" and is poorly measured by the metric. This sort of problem shows the importance of using more than on metric in the analysis of a product (see later in this chapter). Studies in Chapter 7 showing the correlations between metrics show that what cyclomatic complexity might not show as overly complex has other characteristics that are.

Secondly the metric measures the lexical rather than semantic characteristics of software. There is no facility for differentiating between the different structures that might exist within a sub-unit due to nesting and the context of the decision statement. For example, three IF statements nested inside a WHILE statement have the same $\nu(G)$ as the same statements existing sequentially.

The practice of modularizing programs can results in higher $\nu(G)$ for the programs. For example, if a related portion of a program is modularized then the $\nu(G)$ for the program increase by at least two (from the *2p* part of $e-n+2p$). This occurs because while the $\nu(G)$ for the new module's statements should remain the same as for before modularization, there are the extra edges to be added for the call to the module, and for it's return. Measuring the cyclomatic complexity of a program isn't all that useful as the component $\nu(G)$ values for each of the modules in the program are lost.

The metric also has problems with programs written in languages where the flow of control is not well defined. This is because the metric was defined at a time when the predominant programming language was FORTRAN. An example of an undefined flow of control is implicit exception handling in ADA.

These drawbacks have been described by a variety of authors who have suggested

allowances for some of these problems and extensions to the metric[19].

Studies have shown erratic performance in using the metric as a predictor of such qualities as maintainability, understandability, error proneness and the effort for development. It correlates highly with the lines of code measure, indicating in part that they are both measure similar things, and in significant cases the lines of code measure was better at predicting errors[20].

The SMW system includes a cyclomatic complexity tool for gathering metric data from C programs. It was developed to for use in evaluating claims about it's correlation with lines of code, and for comparison with a similar metric, NPATH, described later.

2.3 Specification Product Metrics.

Specification metrics are used to measure the software product produced after the specifications for a software system have been produced. This can occur either during the end of the specification phase or during the early design phases. Typically specification metrics are more concerned with predicting the cost and effort associated with developing a system, so that resources can be allocated for development.

A specification metric, Albrecht's function point analysis, is discussed below.

2.3.1 Function Points

Function point analysis was developed in the late 1970's[21]. It is designed to be used as a predictor of development effort for a software system, but can be adapted to predict characteristics in the final product. Function point analysis is based upon measuring the different functions that occur within a system, and assigning these weights based upon the complexity of that type of function.

The software is divided into five different sorts of functions. These are user inputs, user outputs, user enquiries, master files and external system interfaces. Once this is achieved then the total number of function points is calculated as follows:

---

[19]Myers, G J; An extension ot the cyclomatic measure of program complexity; ACM SIGPLAN Notices; Vol. 12; No. 10; 61-64; 1977

[20]Basili, V R and Perricone, B T; Software Errors and Complexity: An Empirical Investigation; *Communications of the ACM;* Vol. 27; No. 1; 42-52; 1984

[21]Albrecht, A J and Gaffney, J E Jnr; Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation; *IEEE Trans. Software Engineering;* Vol SE-9; No. 6; 639-648; 1983

$$(4*I) + (5*O) + (4*E) + (7*P) + (10*F)$$

where I = number of user inputs

O = number of user outputs

E = number of user enquiries

P = number of external system interfaces

F = number of master files

Each component of the total may be adjusted (±35%) depending on the complexity of the functions being performed and once the total number of function points has been calculated the final total may be increased or decreased by a factor of up to 25%. These adjustments must be applied subjectively by the measurer.

The use of function points depends greatly on the maintenance of a history of other similar projects. Function point totals vary greatly between development environments and application types, so it is not always possible to compare projects directly. By maintaining a history of other similar projects though the measure has better information with which to adjust the function point total by. Function points are typically used in the business information systems area for such applications as on-line banking systems.

Drawbacks with the metric are that the function points are determined manually which includes a level of subjectivity into the measurements, and the adjustment and interpretation of the function point total require familiarity with previous applications of the metric.

The SMW system doesn't include a function point metric, but the system could be coerced into using the data object structures within it's program information schema (Chapter 4) to store information for function point analysis, such as the different input and output objects.

2.3.2 Other Specification Metrics.

Two other specification metrics are the Bang metric proposed by DeMarco[22], and a metric based upon the measurement of a formal specification language, OBJ.

The Bang metric is derived from more formal specification and design notations such as data flow diagrams, data dictionaries and entity-relationship diagrams. Using these notations two metrics are calculated, one to evaluate from the data flow diagram how function oriented the software is, and the other evaluating entity-relationship diagrams in an effort to data oriented it is. From these metrics the overall classification of the software may be determined.

---

[22]DeMarco, T; Controlling Software Projects: Management, Measurement and Estimation; *Yourdon Press, NY, U.S.A*; 1983

The use of design notations is significant as it could be possible to automatically process the notation documents, and to remove some of the subjective analysis. No empirical evidence exists for the metrics use though, and like function points it's use would be best when combined with a historical database.

A metric based on measuring the specification notation OBJ has also been developed[23]. This eliminates subjective measurements, which can introduce ambiguities into the metric data, a formal notation, OBJ. Preliminary studies with small modules has found it to be a possible predictor of cyclomatic complexity and module length based upon the number of statements it takes to describe an operator on an abstract data type.

2.4 Design Metrics

Design metrics offer the measurer the ability to measure characteristics of the software product at an early stage in order for predictions to be made about the quality of the implemented product. Typically design metrics are centered around the structure of the software, in terms of identifying characteristics about the components or modules that make up a piece of software, and the relationships between these components.

Interest in design metrics is high due to the early feedback that they can give about the nature of the design of a piece of software. The importance of this can be seen from the statements made earlier that a large proportion of the non-clerical errors in a software system can be attributed to the design of the software. In spite of the interest in these sort of metrics, design metric usage and development has been hindered in the past by two problems.

The first problem is in the lack of use of standard design notations for describing the design of software. Ideally the collection of metric data should be through the use of an automated system in order to reduce the number of subjective influences, and the informal methods used in the past, and the lack of machine readable design products has prevented the development of automated collection. However with the emergence of Computer-Aided Software Engineering (CASE) tools running on powerful micro-computers and workstations, which allow for the storage of a variety of design notations such as structure charts, data flow diagrams and entity-relationship diagrams, this problem is starting to be addressed. In the next chapter an automated system that processes a machine storable design notation is discussed[24].

The second problem is that there has been little validation of design metrics. It is possible to

---

[23]Samson, W B, Nevill, D G and Dugard, P I; Predictive Software Metrics based on a Formal Specification; *Information and Software Technology;* Vol. 29; No. 5; 242-248; 1987

[24]Ince, D C and Hekmatpour, S; An Approach to Automated Software Design Based on Product metrics; *Software Engineering Journal;* 53-56; 1988

reverse engineer existing software systems to retrieve the design characteristics present in them and then to analyze those results. But given the time it takes to develop large scale software systems, for a design metric to be used in many development projects and results collated, several years may have passed. Thus metrics proposed in the late 1980's may have to wait until the mid-1990's before the data extracted can be used to validate the design metrics.

Design metrics tend to fall into one of three categories. Those that deal with the relationships within modules (intra-module), those that deal with relationships between modules (inter-module), and those that attempt to do both. The following sections will discuss each category briefly with reference to existing design metrics.

2.4.1 Intra-module Metrics.

Intra-module metrics are usually based upon the measurement of cohesion and control structure within a module. Cohesion is a measure of the relationship between the elements of a module with the task that module is to perform[25]. Ideally each module should have a high cohesivity, i.e. it should consist of only elements that are concerned with the module's primary purpose. Any other elements that are occur in the module and perform task other than those that support the primary purpose of the module lower the cohesion of the module.

Intra-module metric suffer from the fact that in order for measurements to be carried out the internals of the module have to be well defined. Normally this only occurs late in the design stage where some program design language, such as schematic pseudocode[26,27], has been used to define the module. These metrics therefore can't give predictions on the characteristics of the software at a stage much earlier than the coding phase of development.

An example of intra-module metrics being used is the Partial Metrics System developed by Reynolds. This system is discussed more fully in the next chapter.

2.4.2 Inter-module Metrics.

Inter-module metrics are concerned with measuring the complexity of the links between modules. This complexity is measured in a variety of ways, from measuring the graph

---

[25]Troy, D A and Zweben, S H; Measuring the Quality of Structured Designs; *J. Systems and Software;* Vol. 2; 113-120; 1981

[26]Robillard, P N; Schematic Pseudocode for Program Constructs and its Computer Automation by Schemacode; *Communications of the ACM;* Vol. 29; No.11; 1072-1089; 1986

[27]Robillard, P N; On the Evolution od Graphical Notations for Program Design; *ACM Sigsoft;* Vol. 14; No. 1; 84-88; 1989

impurity of the structure chart of program[28,29], to measuring the level of coupling between modules, and the flow of control within a system of modules[30]. Research by Troy and Zweben[31] indicates that coupling may be the predominant characteristic within designs that can be used to predict the error-proneness of software. Coupling by their definition is a measure of the strength of association established by the interconnection from one module of a design to another. The degree of coupling depends on how complicated the connections are, and the type of data present in the connections. The more complex the interface between modules, then the more the modules are bound together and changes in one are more likely to affect the other.

In the next chapter an automated design metric tool used to measure the complexity of designs based upon measuring graph impurity is described[32].

A recent design metric is that of McCabe and Butler that extends work previously done by McCabe on cyclomatic complexity. The cyclomatic complexity metric was adapted to measure the complexity of structured designs based upon structure charts that support the use of notation to show iteration and conditional invocation of modules.

Using the structure chart (or design tree as it is referred to) the design is broken down into modules and each module has it's module design complexity, *in(G)*, calculated. The module design complexity is defined as the cyclomatic complexity of the reduced flow graph of the module. The reduction is performed to eliminate any complexity which does not influence the interrelationships between the modules in the design. This is best explained graphically.

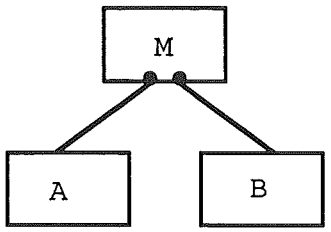


Figure 2.2 Subtree from a module, M.

---

[28]Yin, B H and Winchester, J W; The establishment and use of measures to evaluate the quality of structured designs; Proc. ACM Software Qual. Ass. Workshop, 45-52, 1978

[29]Benyon-Tinker, G; Complexity measures in an evolving large system; Proc. ACM Workshop Quant. Software Models; 117-127; 1979

[30]McCabe, T J and Butler, C W; Design Complexity Measurement and Testing; *Communications of the ACM;* Vol. 32; No.12; 1415-1425; 1989

[31]Troy, D A and Zweben, S H; Measuring the Quality of Structured Designs

[32]Ince, D C and Hekmatpour, S; An Approach to Automated Software Design Based on Product Metrics;

In Figure 2.2, a module M is shown that can conditionally invoke two other modules, A and B.

The flow graph for this module can be seen in Figure 2.3, and using a series of iterative reduction rules the reduced graph shown in Figure 2.4 may be created. The cyclomatic complexity measure of this graph is 3, which becomes it's module design complexity.

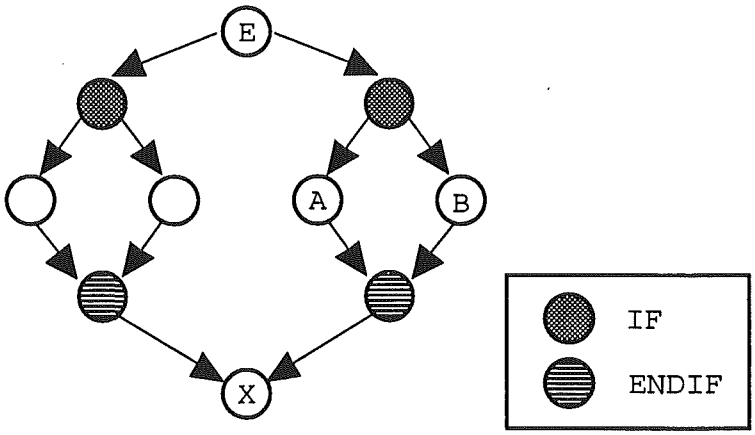


Figure 2.3 Flow graph for module M. (E=entry node, X=exit node)

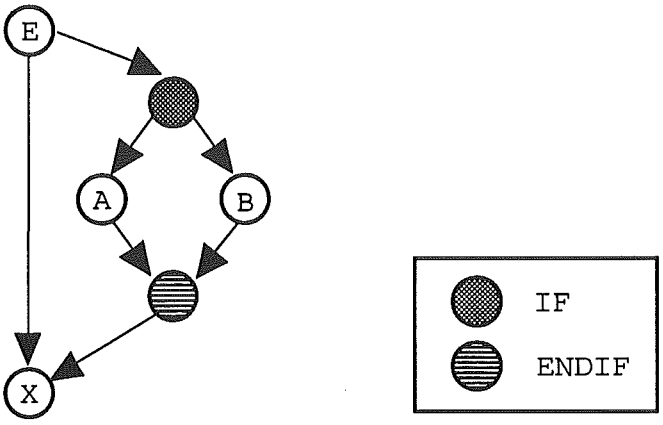


Figure 2.4 Reduced flow graph for module M. (E=entry node, X=exit node)

These module design complexity measures are combined to produced an overall measure of design complexity for an entire structure chart or sub-tree in a structure chart. The design complexity is defined as:

$$S_0 = \sum_{i \in D} iv(G_i)$$

where D is the set of descendent modules of the root module, M, unioned with module M.

The authors of this metric also propose it's use in calculating the integration complexity of a design. This measure allows for prediction of testing effort, and localization of areas in the design that will require a higher proportion of testing when the design is implemented.

2.4.3 Hybrid Metrics - A combination of inter- and intra-module measures.

These metrics attempt to combine the internal complexity and cohesion measures of intra-module metrics, with those measures of coupling and structure complexity of inter-module metrics.

Possibly the most well known of the design metrics, the information flow metric developed by Henry and Kafura, is a metric of this sort. It measures the flow of information between modules by measuring the information fanin and fanout values of a module, based upon the number of calls to and from a module, and the number of data objects passed in and out. It also measures the length of the module in lines of source code (LOC) to get an intra-module metric value. The measures are then combined in the following way to give an overall value for a module.

$$\text{length} * (\text{fanin} * \text{fanout})^2$$

The drawbacks in this metric are those that are associated with intra-module metrics in general, in that to define the length of the module the module must already be coded and the design implemented. Also the module for the information flow between modules is limited to a count of data objects and module calls, with little regard for the complexity of those data objects, so that a complex record structure is treated the same as a simple integer value. Modules that have fanin or fanout values of zero can't have values calculated for them, so that a module that is called by 20 other modules, has a length of 1000 lines of code, and then doesn't call any other modules has a information flow value of 0. The squaring of the fanin/fanout product was determined arbitrarily from empirically examination, and the lack of a definitive length metric could hinder consistent measurement.

Another hybrid design metric has been developed by Card and Agresti[33].

This metric was developed to measure the complexity of the overall design of a software system at a time when the internal structure of it components may not have been defined. The basis for the metric is the measurement of the results of the architectural design of a software system. Architectural design is defined by the authors as the process of partitioning the required functionality and data of a software system into parts that work together to achieve the full mission of the system. The measurement of complexity has deliberately left at an abstract level to allow the measurement of products produced from a variety of design methodologies.

The measurements of the software product are designed to take into account both the

---

[33]Card, D N and Agresti, W W; Measuring Software Design Complexity; *J. Systems and Software;* Vol. 8; 185-197; 1988

complexity of the system as a whole and the complexity that may be incorporated in each component in the system. The complexity measures used are based upon measures of the number of modules in the system, the number if I/O variables (parameters) each module uses and the number of modules called by each module (fanout).

The complexity measure for the entire system design can be broken down into two measurements, a measure of the total inter-module or structural complexity of the system, and a measure of the total intra-module complexity. The structural complexity is based upon the concept of functional decomposition that leads to a hierarchical network of modules within a software system. Each module in the system has a work load based upon it's processing of input and output data, and the work load of a module can be deferred to lower levels by the module calling other modules. This deferring of the work load to other lower modules reduces the local complexity of a module, but through the creation of more modules in the system increases the structural complexity.

Card and Agresti define the total design complexity, $C\sim$, as follows:

$$C\sim \quad = \quad S\sim + L\sim$$

where $S\sim$ is the structural complexity and $L\sim$ is the local complexity.

In order to be able to compare designs these complexity measures were adjusted relative to the number of modules in a system, n, to give

$$C \quad = \quad \frac{C\sim}{n}$$

$$L \quad = \quad \frac{L\sim}{n}$$

$$S \quad = \quad \frac{S\sim}{n}$$

The authors hold the view that modules with the simplest structural complexity are those which call no other modules, that is their fanout values are zero. The structural complexity of a system is based upon the average squared deviation of actual fanout from this simplest structure, and is defined as:

$$S \quad = \quad \frac{\Sigma f_i}{n}$$

where f_i is the fanout value of module i, and n the number of modules in the system.

The local complexity is defined as a measure of the amount of work a module has to perform. This is based upon the number of data items that serve as inputs and outputs to the module is a measure of the amount of work to be performed, and the idea that deferred functionality causes a decrease in the internal complexity of a module. The local complexity

can be defined as:

$$L = \frac{\Sigma \frac{v_i}{f_i + 1}}{n}$$

where f_i is the fanout value of module i, v_i is the number of I/O variables that module i uses and n the number of modules in the system. It is assumed for simplicity's sake that the work load or each module is divided evenly between itself and each of it's sub-ordinate modules, thus the addition of one to the fanout to include the module itself.

The metric was evaluated on eight medium-sized FORTRAN systems totalling about 1000 individual modules. Evaluation was through the subjective analysis of the designs by a senior project manager involved with all eight projects and using the metric to evaluate the error rate in the designs, that is the number of code changes required at the implementation and system testing phases. In the first case the four designs described as "bad" had the highest metric values. In the second evaluation about 60% of the variation in the error rate could be explained by the values of complexity calculated.

While the evaluation of this metric requires much more evaluation it does show a promising direction in trying to incorporate information available earlier in the design phase, in an attempt to unite both intra and inter module complexity into a single model. The measure only captures a sub-set of possible complexity factors and ignores others such as the use of global or common data structures as a method to pass information between modules, but it's serves as a good example of the new sort of design metrics that are being researched.

2.5 Code Metrics

Code metrics measure characteristics of a piece of software product found in the source code product. The "Classic" metrics described earlier are all code metrics, and up until recently these, or variations of those metrics, would have been the only code metrics in widespread usage.

2.5.1 The Place of Code Metrics.

The main problem with code metrics is that by the time you have the software product to measure the characteristics of the software inherent in the design have already been implemented. Thus any feedback that can be obtained about the complexity and quality of the software product is available too late in the development life cycle. Any changes to reduce complexity within the software at this stage require either redesigning whole sub-systems and re-implementing the software, or become part of the on-going maintenance of the software. Another problem with code metrics is that they tend to be language specific. If the software design is implemented in a variety of difference programming languages, such as C with additional hand-coded assembly language routines, then slightly different

collection methods and characteristics need to be used.

Code metrics have been implemented to work with the pseudocode definition of software that can occur within the late design stages of a project[34], and these will provide feedback at a slightly earlier stage but as noted previously much of the errors found in software are attributable to design errors. Also the pseudocode metrics are based upon the "Classic" metrics and as such inherit their flaws.

Perhaps the biggest areas for the application of code metrics though are the fields of software testing and maintenance[35]. In the first field code metrics may provide feedback on the complexity of the coded product that would provide a good starting place for testing. This was suggest by McCabe when he developed the cyclomatic complexity metric[36]. Modules and functions with high complexity measures and sizes could be identified and more resources, such as time and people assigned to the testing of those functions. In the area of maintenance code metrics could be used to identify those modules in a software product that could be in need of rewriting, or be used to predict the effect of a enhancement or cost that maintenance of a particular module might be compared to other modules.

The SMW system allows for the identification of modules by metric values. Thus it could supply information on all the modules that fall in the top 25% of a particular metric, or exceed at least two of three specified metric value thresholds. This facility to do queries like the above ones is discussed in Chapters 4, 5 and 6.

2.5.2 NPATH - A Relatively New Code Metric.

One recently developed code metric is the NPATH metric[37]. The NPATH metric was developed to overcome problems with the cyclomatic complexity metric in measuring the control flow complexity of a program sub-unit. The NPATH metric measures the number of acyclic execution paths within a C function. The metric is specific to the C language although should not be too hard to adapt to a similarly structured programming language.

The specific problems that NPATH is designed to rectify are:

• The cyclomatic complexity only represents the fundamental number of circuits in the flow

---

[34]Reynolds, R G; The Partial Metrics System: A Tool to Support the Metrics Driven Design of Pseudocode Programs; *J. Systems and Software;* Vol. 9; 287-295; 1989

[35]Leach, R J; Software Metrics and Software Maintenance; *Software Maintenance: Research and Practice;* Vol. 2; 133-142; 1990

[36]McCabe, T J; A Complexity Measure

[37]Nejmeh, B A; NPATH: A Measure of Execution Path Complexity and its Applications; *Communications of the ACM;* Vol. 31; No. 2; 188-200; 1988

graph, but in fact this is only true when the predicate node in the flow graph has an outdegree of 2.[38]

• McCabe's measure does not distinguish between different sorts of control flow structures, some of which the author of NPATH argues are harder than others to use.

• The NPATH metric takes into account the nesting of control structures within each other.

The NPATH metric is suggested to be used in the following ways.

• Selecting functions for walk-through/inspection. Functions with a high NPATH values (the top 25%) should be isolated for closer inspection.

• Allocation of functional testing resources. Rank functions in order of NPATH values and then assign resources based upon that ranking. Thus the testing resources are proportional to the number of acyclic execution paths through the function.

• Defining module design criteria. NPATH can work with program design languages that use the C control structures, and could be used to identify functions earlier that might be in need redesigning.

The calculation criteria for the NPATH metric are given in Appendix C.

2.6 Trends and Directions in Software Product Metrics.

In the last fifteen years or so software metrics have undergone several major trends. The most major of these trends is the switch in direction from analysis of the source code product of a piece of software to analysis of the specification and system design[39]. Other trends include the development of better tools and methodologies for facilitating the automatic collection of metric data, a change in the analysis methods of metric data, and the moves to establish a better set of rules and models with which to better describe metrics and measurement philosophies.

In a study carried out of metrics literature in 1986 the breakdown of the classes of metric literature published was described[40]. The distribution of the literature for different classes of metrics can be seen in Figure 2.5. From the graph it is apparent that even though the types of literature include process metrics, such as method and software quality assurance (SQA)

---

[38]Evangelist, M; An analysis of control flow complexity; *Proc IEEE COMPSAC 84*; 388-396;1984

[39]Shepperd, M; An Evaluation of Software Product Metrics; *Information and Software Technology;* Vol. 30; No. 3; 177-188; 1988b

[40]Cote, V, Bourque, P, Oligny, S and Rivard, N; Software Metrics: An Overview of Recent Results; *J. Systems and Software;* Vol. 8; 121-131; 1988

metrics, as well as three different classes of product metrics, that code metrics are most dominant in the literature.

If only product metrics are included in the breakdown of literature (Figure 2.6) then it becomes even more apparent that code metrics have at least up until 1986 dominated the filed of software product metrics.

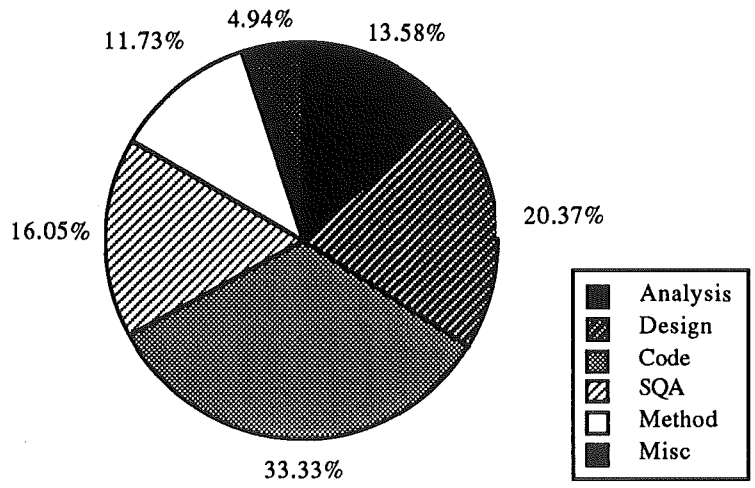


Figure 2.5 Distribution of Software Metrics Literature (1986).

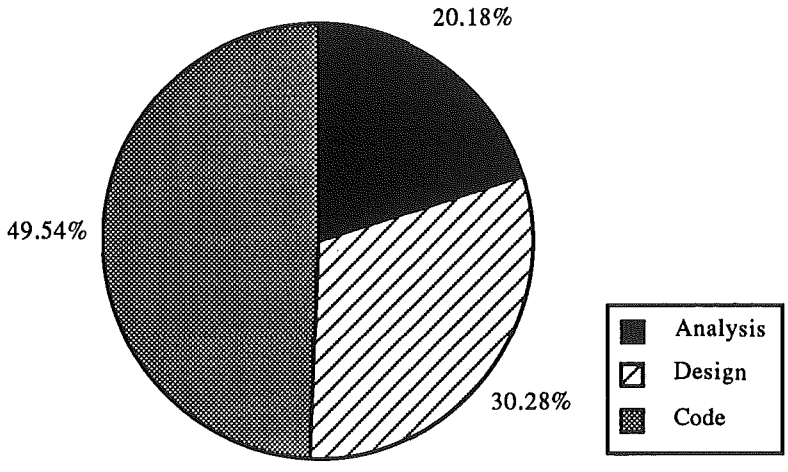


Figure 2.6 Distribution of Product Metric Literature (1986).

However in the last few years the emphasis has shifted away from code metrics, except for special cases such as testing and maintenance, and towards the more abstract metrics based on analyzing software design and specification. This has occurred because metrics are being

used more to predict the quality and characteristics of software, as opposed to actually describing the already coded product. This provides feedback about the software product to the software developer at an earlier stage in the development process which can allow error prediction and isolation of parts of the design for examination. Also the models that design metrics have been built upon appear to be more in line with the current models used in software development[41]. Any improvement in the ability to predict software characteristics earlier is to be investigated given the importance of design upon the quality of final software product.

The trend towards the automation of software metrics collection has been driven for by a desire for more objective data, and the ability for metrics data collection to be carried out without intruding noticably in the development process. As noted previously, this trend has been aided by the availability of software that supports the storage of design notations in a form that is able to be processed automatically. The next chapter describes a variety of automated metrics collection tools and methodologies that have been developed.

Another trend in the software metrics field has been the increased discussion on the analysis of metric data. Critiques of earlier metric experiments have been published[42], as well as general papers on metric data analysis[43,44], and application of new analysis techniques to metric data, such as factor[45] and outlier analyses[46,47], which rely on the distribution of the data rather than a fixed upper threshold value.

The last major trend in software metrics to be mentioned is the development of new philosophies and models to define software measurement, and to provide a coherent underlying base from which metric research and analysis can be carried out. Several ideas on these aims and some basic frameworks have been defined[48,49]. One idea to come out of

---

[41]Shepperd, M; An Evaluation of Software Product Metrics

[42]Fenton, N and Melton, A; Deriving Structurally Based Software Measures; *J. Systems Software;* Vol. 12; 177-187; 1990

[43]Pickard, L M; Analysis of Software Metrics; *Proc. Centre for Software Reliability Conference: Measurement for Software Control and Assurance;* 155-180; 1987

[44]Myrvoid, A; Data Analysis for Software Metrics; *J. Systems Software;* Vol. 12; 271-275; 1990

[45]Coupal, D and Robillard, P N; Factor Analysis of Source Code Metrics; *J. Systems Software;* Vol. 12; 263-269; 1990

[46]Kitchenham, B A and Linkman, S J; Design Metrics in Practice; *Information and Software Technology;* Vol. 32; No. 4; 304-310; 1990

[47]Shepperd, M and Ince, D; Metrics, Outlier Analysis and the Software Design Process; *Information and Software Technology;* Vol. 31; No. 2; 91-98; 1989

[48]Baker, A L, Bieman, J M, Fenton, N, Gustafson, D A, Melton, A C and Whitty, R; A Philosophy for Software Measurement; *J. Systems Software;* Vol. 12; 277-281; 1990

this trend is that there is no one general-purpose metric that can be used, but rather a collection of metrics measuring a variety of characteristics need to used in conjunction with one another.

The increase in information on metric data analysis, combined with improved automated tools and better frameworks and models for describing the results of the analyses should help to overcome the main problem with software product metrics, in that the amount of empirical data for design and specification metrics should improve. This will lead to better validation of metric models, and hopefully better software.

The SMW system incorporates several of these trends in being an automated system that can store program and metric data from the design and coding phases of development, while providing powerful data manipulation tools for data analysis. It is hoped that it will be able to be used to provide test sets of empirical data for metric validation and investigation of metric models, as well as a variety of metric analysis functions.

---

[49]Fenton, N E; Software Metrics: Theory, Tools and Validation; *Software Engineering Journal;* 65-78; 1990

Chapter 3. Automating Software Metrics: Methods and Tools.

3.1 Introduction.

The purpose of this chapter is to give overviews of specific methodologies and tools that have been developed in order to automate or to aid in the automation of software metric data collection and analysis. The tools and methods discussed here by no means describe all the different approaches to the automation of software product metrics, but have been included to show the variety of systems and ideas that have been proposed, and the common underlying themes that influenced the development of the SMW system.

The most common automated tools likely to be encountered are simple tools that, in general, will calculate code metric values for the 'Classic' metrics, i.e. Halstead, McCabe and LOC, for a specific source language, and provide no rationale for the interpretation of the values calculated, nor underline the difficulties and drawbacks in using metrics. Tools for measuring the software product at different stages in the software development process, such as design and specification phases are are not commonly available, and if needed then have to be developed "in-house" by the developer.

3.2 Descriptions of Tools and Methodologies.

The following descriptions are about systems and concepts that occur at design, implementation and maintenance phases of the development process.

MAE - A Syntactic Metric Analysis Environment[50].

SMDC - Software Metrics Data Collection and Analysis system[51].

Software Metrics and Integrated Project Support Environments[52].

Integrating Metrics into a Large-Scale Software Development Environment[53].

The Partial Metrics System[54,55].

---

[50]Harrison, W; MAE: A Syntactic Metric Analysis Environment; *J. Systems and Software;* Vol. 8; 57-62; 1988

[51]Yu, T J, Nejmeh, H E, Dunsmore, H E and Shen, V Y; SMDC: An Interactive Software Metrics Data Collection and Analysis System; *J. Systems and Software*; Vol. 8; 39-46; 1988

[52]Kitchenham, B A and McDermid, J A; Software Metrics and Integrated Project Support Environments; *Software Engineering Journal;* 58-64; 1986

[53]Henry, S and Lewis, J; Integrating Metrics into a Large-scale Software Development Environment; *J. Systems Software;* Vol. 13; 89-95; 1990

[54]Reynolds, R G; Metric-based Reasoning about Pseudocode Design in the Partial Metrics System; *Information and Software Technology;* Vol. 29; No. 9; 497-502; 1987a

[55]Reynolds, R G; The Partial Metrics System: Modeling the Stepwise Refinement Process using Partial Metrics

An Approach to Automated Software Design Based on Product Metrics[56].

3.2.1 MAE - A Syntactic Metric Analysis Environment.

MAE (Metric Analysis Environment) is a software system that through the use of a commercial DBMS and a static analysis tool provides an environment for the analysis of syntactic metrics for software written in COBOL.

Syntactic metrics, as defined by the creators of the MAE system, are measures of software complexity based upon the syntactic characteristics of a software product – in this case COBOL source code. Each complexity metric can be viewed as the mapping m such that:

$$m(C1,C2 \ldots,Cn) \rightarrow \text{measure of complexity}$$

where Ci is the degree to which a particular charateristc exists in the software product.

The syntactic characteristics are stored in the database and the DBMS's query language is used to implement the desired mappings, which in this case include Halstead's Software Science, McCabe's Cyclomatic Complexity, and Lines of Code.

The system has a database schema designed, not unreasonably, for the storage of the components that make up simple COBOL programs and the relationships between them. The static analysis tool parses the COBOL source code and the stores the different syntactic elements it finds in the database.

The database schema is shown below (Figure 3.1) and shows the syntactic elements that the COBOL programs can be divided into, i.e. identifiers, lines, operators, and paragraphs.

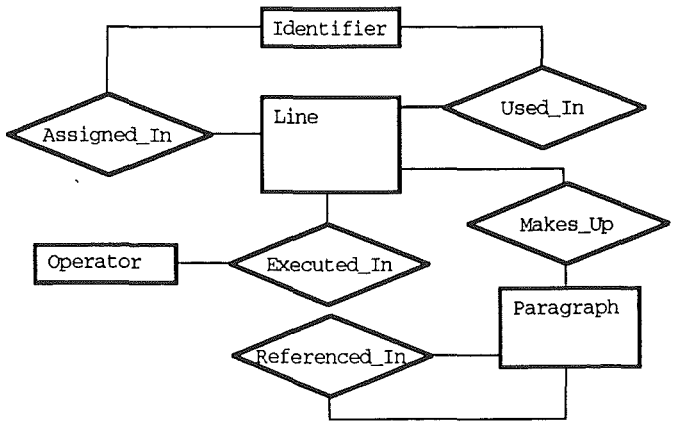


Figure 3.1 Database Scheme for M.A.E.

Using the DBMS, views onto the data can be provided that allow users to access the data for

[56]Ince, D C and Hekmatpour, S; An Approach to Automated Software Design Based on Product metrics

analysis purposes without actually examining the source code. While this can distort the analysis process by removing contextual information, it can allow analysis and sharing of sensitive data with individuals who might not be able to use it otherwise.

The aims that guided the design of the MAE system were:

1. Easy evaluation of a number of metric mappings for a particular database.

2. The environment must be able to support new metrics, and modifications to existing metrics. These should then be able to be reapplied to exising data.

3. The environment should encourage the sharing of data amoung software metrics researchers.

By basing the system on top of a database system these aims have been fulfilled. The query language component of the database facilitates the development of the desired metric mappings, while the storage of the desired syntactic characteristics of the program in the database means that new metric mappings can be developed and existing ones changed, and then evaluated without having to reanalyse the source code. The third aim is achieved by allowing the data to be accessed in such a way that the syntactic characteristics of the program exist but the actual program itself doesn't, allowing data from sensitive or propriety source code to be shared amoung researchers. The MAE system also supports a menu driven interface to facilitate the use of the system, increasing the accessibility of the system in general to a range of users.

The MAE system has several disadvantages.

Firstly each program must have it's own database. Thus multiple versions of the same program at different levels of development would all have their own databases, using more resources that if they could all share the same database. The system currently doesn't support external programs, making it necessary to have only one program source file for the program. Also the system has no provision for tracking control flow within a program block, as only the syntactic values of the IF statements are captured, not their context within the program. Finally, to adapt the system to a new programming language would involve designing a totally new database schema for the new syntactic elements.

3.2.2 SMDC - Software Metrics Data Collection and Analysis System.

The Software Metrics Data Collection (SMDC) system was developed at Purdue University with the aim of providing researchers with a data collection system to investigate the applications of new and existing software metrics. The system is an interactive menu driven system that contains online documentation, and contains metrics data for hundreds of

programs ranging in size from small experimental programs to programs over one million lines in size.

The way that the SMDC system was to promote the aim above was to gather data related to software development in a central location and provide mathematical and statistical functions for manipulating that data.

Data collected in SMDC came from a variety of sources. Data was collected from public domain data, industry supplied data, industry collected data, university experiments and computer libraries. Data of the industry collected type was the most useful, although the time to understand the development process and the data collected took longer than the other types of data. Disadvantages with the other sorts of data include the small size and artificial development environment from university experiments, the lack of choice in data supplied in industry supplied data, and the lack of validation of data supplied from public domain sources. Also each source had it's own way of measuring data therefore difficult to maintain a uniform definition for each software metric in the database.

The data is stored in a set of APL workspaces. Workspaces that contain data that came from similar environments, such as a set of university experiments, are grouped together in the same directory. Each metric is given a unique variable name in each workspace and can be accessed and manipulated by APL. The data files are also available as ASCII text files for users who can't use APL to manipulate the data.

The metrics that the SMDC system supports include lines of code, development effort (person months and hours), duration (calendar time during which development effort proceeded without interruption), Halstead's Software Science, McCabe's Cyclomatic Complexity, and others such as programmer experience.

The SMDC system also support a variety of functions for the analysis of stored data including:

1. Basic statistical functions (mean, var, correlation coefficient, moving average)

2. Statistical test functions (t-test, ANOVA (analysis of var), analysis of covariaence (ANCOVA), Kruskal-Wallis one-way analysis of variance).

3. Scatter plots with multiple independent variables.

4. Scatter plots with regression lines.

5. Histograms.

6. Formatted text reports.

3.2.3 Software Metrics and Integrated Project Support Environments.

Typically software development is with the aid of a few selected tools, such as a compiler, an editor and possibly a debugger, controlled at the operating system level. Integrated program support environments (IPSEs) aim to provide an environment for software development by providing the developer with facilities at a higher level than the operating system. Most commonly IPSEs provide a database that information pertaining to a certain project or projects is recorded in. This information is normally concerned with the technical aspects of the software, such as location of source and object files, and the interdependencies between files, and is aimed at providing the developer with more control over the project.

Kitchenham and McDermid argue that the inclusion of software metrics in IPSEs not only gives the developer improved control over the development process through increased project visibility, and the use of metrics to predict software component characteristics, but that the inclusion of metrics in the environment provides the opportunity to gather information on the effectiveness of the development methodologies being used, and of the utility of the IPSE itself.

The system described by Kitchenham and McDermid is a data collection system that has used for several years in the VME project group at ICL. The system is partially automated with human intervention required for valid data to be captured. The system is based upon the concept of a history records, a special sort of comment, placed into the source code whenever that source file is modified. The history record contains information such as the size of the change in lines of non-comment, non-whitespace code, the reason for, the nature of, and the date of the modification. The history records are automatically extracted and the information is stored in a human readable form so that data can be validated by a user.

The collection of the history record data allows the developer to see that rate of change of modules, the results of module testing, and gives reasons modules to be retested through querying the database for anomalies in the data.

Kitchenham and McDermid identify three main functions of an IPSE's measurement system that should defined.

These are:

1. What metrics should be used.

2. How should the metrics be gathered.

3. What provision is there for the analysis of the metrics.

In the case of what metrics to use the developer should look at the stages of the development process that need to be measured, such as design or coding, as well as what characteristics need to be predicted or monitored. With the collection of the metrics automatic collection is the most objective and the only method feasible for large projects, but the collection methods should be able to cope with changing measurement requirements. Also, while it may be enough to provide the user with an interface to the data, adoption and usage of the system can be encouraged with the provision of tools to perform analysis of the collected data.

Problems and issues identified during the use of the system were:

1. The availability of measurement data changes the requirements of the IPSE users, and will result in new or modified measurement requirements, and changes in the development process.

2. Data captured must be meaningful and able to be analysed by the user.

3. The data collection process shouldn't perturb the development process (even though the authors describe a system where special comments need to placed into the program code).

4. Automatic information capture can be from automatic processes that are invisible to the user, such as monitoring what files are edited and when, or are special tools that the user can request to be run in a similar way to a compiler or revision control system.

5. Information given by users (especially those under stress) may not be accurate.

6. Users behave differently if they perceive they are being monitored, and information obtained from measurement should be helpful to the user.

3.2.4 Integrating Metrics into a Large-Scale Software Development Environment.

This section describes an experiment to produce an automated software tool that could be integrated into a large scale software development environment without disrupting the development process. The tool developers main premise was that while software products may conform to the functional specifications, this doesn't guarantee good software in terms of reliability and other quality factors such as maintainability.

The tool was to be implemented in such a way as to provide the developers with access to the measurement information at various stages of the implementation phase of the development process. The model of the implementation cycle used can be seen in Figure 3.2. This is a subset of the model of the software development shown in Chapter 1 (Figure 1.1), with the design, coding and testing phases decomposed into sub-units.

The metrics were to be used during the implementation phase to indicate error proneness and

when redesign effort might be beneficial. The tool was integrated into a large scale development project that was frozen mid-point through the implementation phase in order to provide validation of the tool.

The tool requirements were:

1. The tool was to be automated to provide objective data.

2. Operation was to be unobtrusive.

3. The tool was to provide threshold values of metrics to aid interpretation.

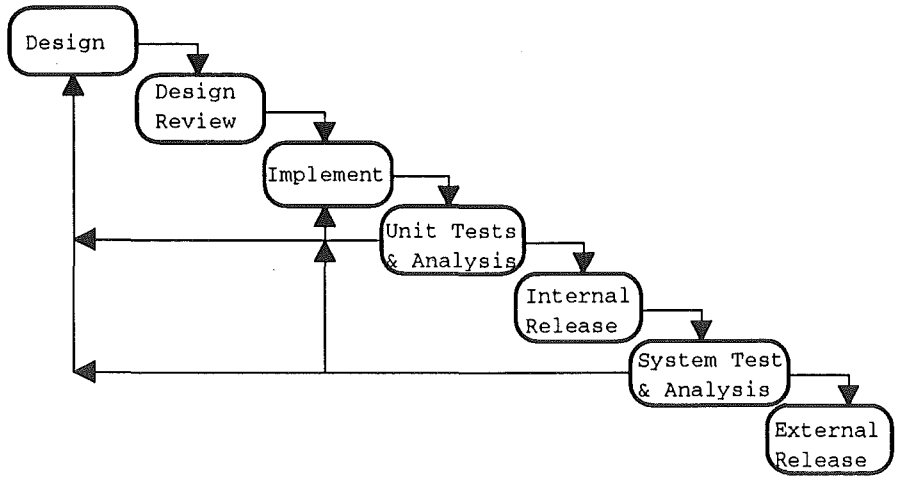


Figure 3.2 Implementation life cycle of software development.

Collection of metrics occurred at different level in the implementation phase. Only metrics that were totally automatable were used as the project was too big for subjective analysis and data was gathered from static analysis of source code. Data was collected at procedure level, module level (small collection of procedures) and at sub-system level (collection of modules).

At the procedure level code the code metrics lines of code, Halstead's Software Science, and McCabe's Cyclomatic Complexity were used.

At the module level the structure metrics Henry and Kafura's Information Flow metric and Belady's Cluster metric[57] were used, as well as a hybrid metric developed by Woodfield.

At sub-system level defect reports were used.

The strategy used was to have individual programmers test the procedures they developed with

[57]Belady, L A and Evangelesti, C J; System Partitioning and It's Measure; *Systems Software*; Vol. 2; 23-39; 1981

code metrics. Then when those procedures were implemented into a module then structure metrics were used to analyses the module. Finally error reports about defects in the product were used to measure the sub-system.

The metric analyser had two parts to it. The code metric analyser analyses the procedure level source code and produces metric values for each procedures, as well as statistics reports on the procedures.

The structure information comes from an existing tool (Communications Database) and provides data that identifies communication lines between modules.

The statistical reports contain information about outlier metric values, as well as alerting the user to metric values that lie outside of accepted thresholds in more than one metric.

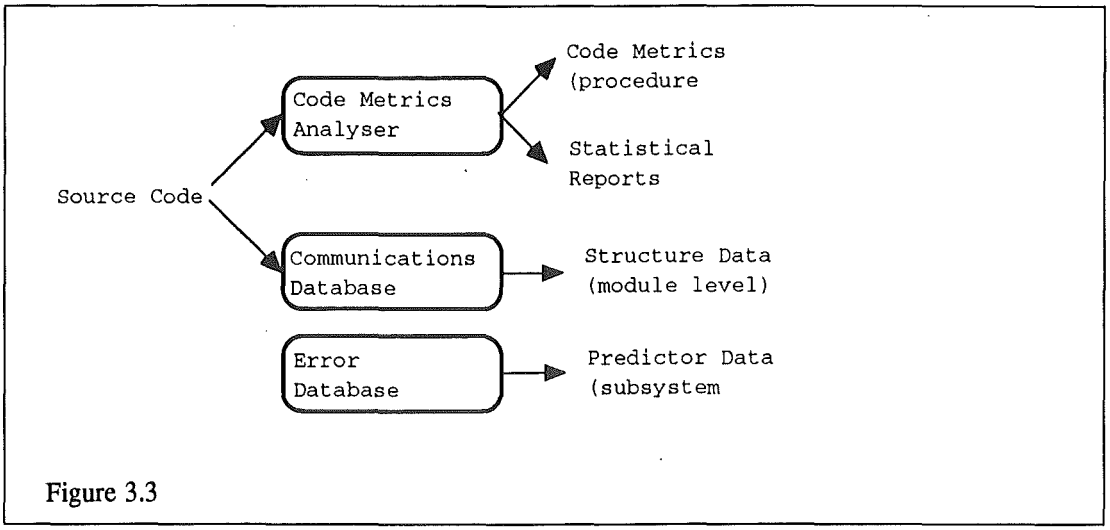


Figure 3.3

3.2.5 The Partial Metrics System.

This section is a description of a software tool that supports metrics-driven design of pseudocode program modules, specifically an ADA based pseudocode, with each stage of the refinement process metrics being assessed in terms of a partial metrics set. This is a methodology to be applied at the later stages of design, and specifically for intra-module design. This occurs after the software system has been partitioned into functional units, (modules) with defined purposes and the internal of the modules are being specified (how it will do things).

The development is based upon the view of top-down program development using stepwise refinement. Stepwise refinement is taken to be the application of successive decomposition of a structured program, with resulting low structures which are series of cascaded and nested repetition and decision constructs. The upper levels of decomposition are design methods such as structure charts and high level flow charts, and the lower would be code level pseudocode.

Partial Metrics were designed to record the contribution of implicit or inferred design decisions to the structural complexity of pseudocode programs. This differs from the traditional approach to metrics which are assessed from completed code and which needed to deal only with the explicit decision made.

When making high level decisions in the design of a module the details pertaining to those decisions may be unknown. For example, the programmer may decide to call a function or perform an arithmetic expression at a certain point in the pseudocode but can not yet express it in detail yet. He or she will insert a stub (a projected component) at this point to be replaced later when the code that performs the task (the prescribed component) is substituted for it. Partial metrics take into account the fact that the designer often knows something about the nature of the code the stub represents, and uses an estimated value for the type of stub added. Stubs that have partial metric values are:

DICT_ATOR	operator
DICT_RAND	operand
DICT_STMT	statement
DICT_PROC	procedure
DICT_FUNCT	function
DICT_FUNCT	Ada package
DICT_TASK	Ada task
DICT_EXPR	arithmetic expression
EXTERNAL	external reference

An example of the use of partial metrics is the calculation of McCabe's Cyclomatic Complexity. Assuming that the conditional branches are two way then:

$$v(G) = \#\text{ prescribed conditions} + \#\text{ projected conditions} + 1;$$

The structure of the Partial Metric System is shown in Figure 3.4. It was designed to support on-line development of larger scale software systems by teams of programmers. Before the system can be used the high level refinement of proposed system into modules has to be complete.

User interacts with the system through the interface sub-system, which performs a security check on the user, allowing access to selected modules. Existing modules are retrieved by the information management system for update, and new modules and changed ones are submitted through the same system. The partial metrics sub-system also checks syntax of the pseudocode, which the authors claim encourages users to make use the system.

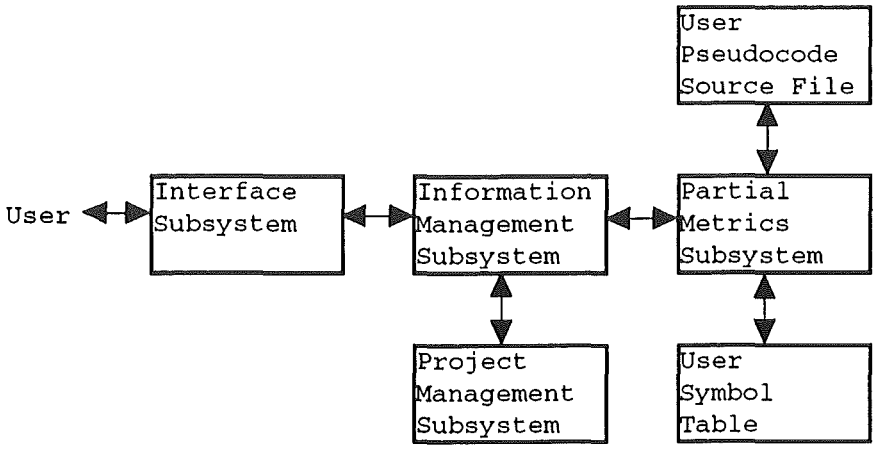


Figure 3.4 The Partial Metrics System.

After submission the user can request the partial metrics sub-system to compute and save the partial metrics values for the new refinement. Based upon those measurement the user can make analyse the effect of the changes upon the complexity of the module. The saved data can be viewed in a variety of plot forms and statistical information is available for each refinement of the modules and system. A variety of metrics are available including Halstead's Software Science and McCabe's Cyclomatic Complexity.

The project management system supports assessment of whole project group and provides summaries of the metrics for the set of modules in the project over the duration of the project.

3.2.6 An Approach to Automated Software Design Based on Product Metrics.

The approach described in this section is very different from the other methods and systems described in this chapter in many ways. Firstly it is concerned only with information available in the design phase of the development process, secondly it is concerned with inter-module relationships, and thirdly it uses operational research techniques in the form of an objective function.

The approach is based upon two concepts.

- That design is a process of examining alternatives and there is probably no single design for a software system. Rather there is a solution space of designs, all of which satisfy the functional specifications of the program, and the developer should choose the best of those designs. In selecting the appropriate design, software metrics should be used to aid the developer's decision.

- That the underlying concept that motivates research in design metrics is the concept of

modularity within a software system, whether it is concerned with information flow or with change impact, and that the bases for system design metrics are connectivity and modularity.

Using these concepts Ince and Hekmatpour identify three areas of system design metrics research concerning design tools.

1. Tools which process one system design and produce values of system complexity.

2. Tools which process a representation of the design solution space and select a design within that space.

3. Tools which process a functional specification of a software system and then synthesise the design solution space, and select a design from within that solution space.

The tool developed is one of the second area, in that it takes the representation of the design solution space and then selects the best design from it that meets the criteria specified by the developer. The solution space is represented in the form of an AND/OR graph, and the best design is the one which minimises the graph impurity of the design's structure chart. The graph impurity is a measure of how tree-like a graph structure is.

The AND/OR graph is similar to a structure chart. Each node represents a program unit such as a procedure, module or database (file, memory based data structure). An edge joining two nodes represents either a module calling another module, or a module using a database. Edges from a module that are joined by an arc represent an AND decision while no arc between edges represents an OR decision.

In the example shown in Figure 3.5 the graph represents:

> A, together with B and C, or with D. If D is selected then the design will also include E or F or G.

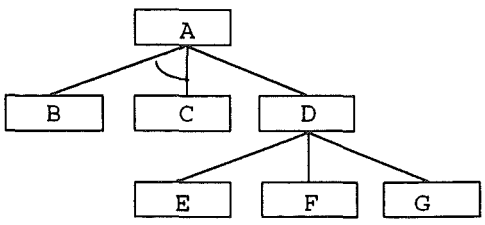


Figure 3.5 Example of an AND/OR Graph.

Using this notation the entire system design solution space can be represented.

Graph impurity is a measure of how much a graph diverges from a pure tree structure. The

higher the graph impurity, the more complex the graph is and the "worse" the design represented by the graph. In Figure 3.6 graph A has a lower graph impurity that graph B.

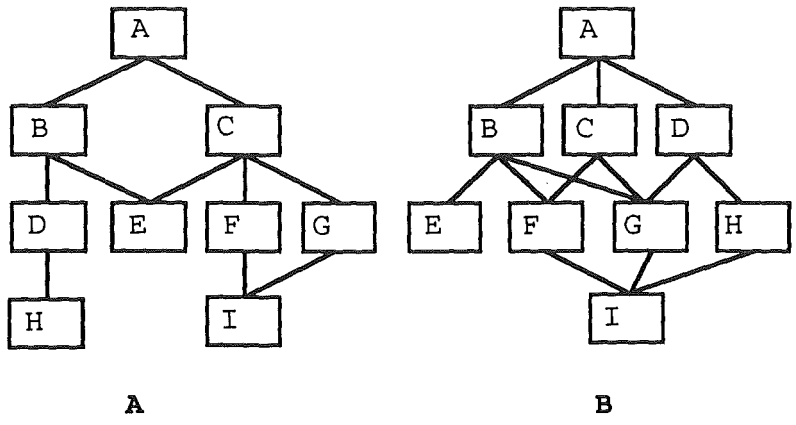


Figure 3.6

The justification is, that for maintenance, a graph-like structure is worse that a tree-like structure, as the levels of coupling in the system will increase. This should be tempered with the fact that a well designed system may use subroutines to save space for code that is frequently used.

The optimal designs in the solution space can be produced by creating an objective function based on the developers criteria, for example size of design and graph impurity, and then minimising that function.

In a system that is being maintained and updated constantly, then the complexity of the system will be of greater importance than the size. However in an embedded controller the size of the design might be the critical factor and the developer would adjust the objective function to cope with this.

The software tool developed by Ince and Hekmatpour takes it's input from the user either in the form of an augmented structure charts, or in the form of a file containing the graph in the form of a module interconnection language. The input is converted into an internal representation and the design is optimised using the objective function with branch-and-bound techniques. The optimal design is then fed back to the user.

Due to the intensive nature of the optimising the largest systems possible for analysis in reasonable time frames can contain no more than 100 decisions and 1500 modules, making it useful for medium sized systems or sub-systems.

Also the tool allows the user to select optimal designs for a system that is evolving, or the user can alter the objective function to see what the designs would be like if circumstances changed.

This allows the designer to see the effect that maintenance might have, and to cater for fuzziness in the design specifications.

3.2.7 Issues Raised by the Tools and Methodologies.

There are several issues that are current throughout the automated systems discussed. The main issue is that the metrics collection process should not interfere unduly with the software development process. Some systems such as PMS were implemented in such a way as to have the development and metric analysis environments existing as a single environment, where as others such as the MAE system capture data from the software product that will be used in the metric analysis at a later date. These help to minimize the perturbation of the development process by hiding the actual metric analysis task from the developer, but methods such as having to explicitly place comments in the software product manually during development add an extra responsibility and work load on the developer.

Another issue is that the data collected should be objective, and that the best way to do this is to automate the metrics collection process. Thus each type of metric data is calculated or collected by the same automated tool each time, which makes sure that all the data corresponds to the same definition of the metric.

The metric data, once collected, should be able to be used. It should be able to be easily obtained, manipulated and analyzed and presented in a form that is usable. Also it is desirable to be able to adapt and adjust the metric collection tools as the requirements for metric data change, and as the metric data is analyzed and deficiencies in the measurements discovered.

The SMW system aims to address these issues. Program information can be stored in the SMW system to minimise the disruption caused by directly accessing the software product at analysis time, as well as storing the results of analyses to prevent multiple accesses to the product each time the metric data is required. The collection of metric data is done through the use of automated tools based upon consistent metric definitions. And the metric and program data is stored in an environment that promotes the manipulation of the information for analysis and reporting purposes, as well as providing some facilities for those purposes.

Chapter 4. The Design of the SMW System.

4.1 Introduction

This chapter contains a discussion of the design of the SMW system. The actual implementation of the SMW system and it's components are discussed in the following two chapters, where issues pertaining to the development tools used, the physical structure of SMW, and the issues raised in implementing the design are described.

In discussing the design of the SMW system the aims of the system as a whole must firstly be examined and then design goals developed from those aims. These design goals should take into account the issues raised by other work done in this field, particularly that of automating metrics data collection and analysis as described in the previous chapter, as well as some of the limitations due to the resources available.

After those design goals and considerations have been described then the actual design characteristics that were decided upon will be shown to form a coherent system that achieves, at least conceptually, what the SMW system intends to provide to it's users and to the field of software metrics.

4.2 Design Goals.

In the introduction to this thesis (Chapter 1) the aims of the SMW system were put into perspective by the problems facing the software developer and researcher at the current time. Specifically the SMW system aims to provide the software developer and researcher with a flexible, extensible and powerful environment for the capture, maintenance and analysis of software metric data.

Such an environment would have the following characteristics:

• Flexibility in terms of being able to provide the user with a wide range of data for metric analysis, and also to provide methods of manipulating the stored data.

• Extensibility in terms of facilitating the development of new tools for analysis of stored data and capturing of new data.

• Provision of a basic set of tools for the maintenance of data in the SMW system, ranging from the capture of new data to maintaining the integrity, and even removing data already stored in the system.

• Provision of resources to provide researcher with basic analysis facilities, such as basic statistical functions, and graphical display of data.

• Facilities to enable the archiving of the data used for analysis, along with the results of the any analysis performed.

As well the automated metric systems studied in the previous chapter raised issues which should be taken into the SMW system's design.

4.3 SMW Components

In order to provide the functionality required by the design goals it was necessary to partition the SMW into several types of components (Figure 4.1). These components are:

• The SMW Data Repository or Database.

• The SMW Data Interface.

• The SMW Data Collection and Manipulation Tools.

The SMW database provides a centralized storage location for the data that the SMW system is to deal with. It contains information on program characteristics and an archive of software metric values.

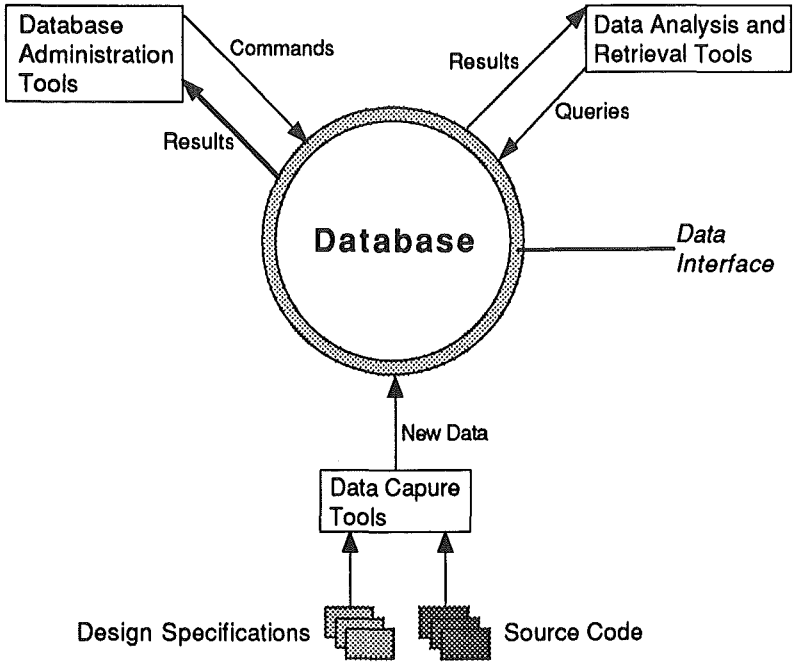


Figure 4.1 The SMW Components.

The partitioning scheme is designed to work as follows. The central database provides a location that is a totally separate entity from the development process. It contains information gathered by data collection tools and manipulated by data manipulation tools. The only way to access the data inside the database is through the data interface, which provides consistent

methods of accessing and manipulating the data. Thus every tool created will access the database through the same methods, thus facilitating the ease of access (each tool works similarly) while restricting methods of access to aid in maintaining the integrity of the data in the database, and the validity of data manipulated.

Typically the only interaction between the development products and the SMW system would be through specialized data collection tools that would each perform a single well-defined task (for example, calculating a particular code metric).

4.4 The SMW database.

The SMW database is the central component of the SMW system and it's purpose is two-fold. Firstly the database must be able to store characteristics of programs that are needed for analysis and secondly the database serves as an archive of results generated by analysis.

The storage of program data or characteristics of programs that are useful for calculating metrics has been discussed in the previous chapter with reference to the MAE system[58]. The main benefit that can be derived from storing program characteristics is that it helps to limit the intervention in the development cycle by minimizing access to the software products. Once that program characteristics have been captured by analyzing the product and then placed in the database, no further access to the software products may be needed, but instead all further data analysis can be carried out by accessing the data in the database.

A second benefit is that the data in the database can be aliased as it's captured to to hide information in the program version that may be sensitive, either in a commercial or research sense. Data is still able to be selected from the database through the data interface for analysis but with characteristics of the program which are sensitive being identified in such a way as to preserve the security of the data, for example all data objects and modules being identified by integers rather than by name.

The other objective of the SMW database is to provide a central repository of metric values that have been calculated for programs stored in the database. As a product evolves it's history can be traced, test sets of metric data can be built up. Also metrics can be composite metrics based upon metrics already calculated. For example, size, fanin and fanout metrics can be incorporated into another metric such as information flow. If product metrics are archived in the database then they can also be used a predictors of the complexity and size of future software systems of a similar nature.

[58]Harrison, W; MAE: A Syntactic Metric Analysis Environment

4.4.1 Storage of Program Data.

The storage of program data in the SMW database raised the issue of what program characteristics should be stored in the database. The following objectives were used to guide the selection of what types of data to store.

• The SMW system should be able to be used at different phases in the development process.

• The software products that the SMW system can store information about should not be limited to product described in only one way, for example products coded in C.

• The SMW system should be able to store information about more than one program,, and more than one version of each program.

The first of these design constraints, that the SMW system should be useful in as many of the software development cycle phases as possible (Figure 4.2), meant that the database should be built upon some sort of structure made up of the entities that are common throughout most or all of the development process. For example, there will be some common entities that will be able to be identified in both a program's design, and later when it is being maintained. These entities tend to be related to a program's structure information (such as coupling, cohesion, information flow, binding) rather than with the syntactic characteristics of a particular language.

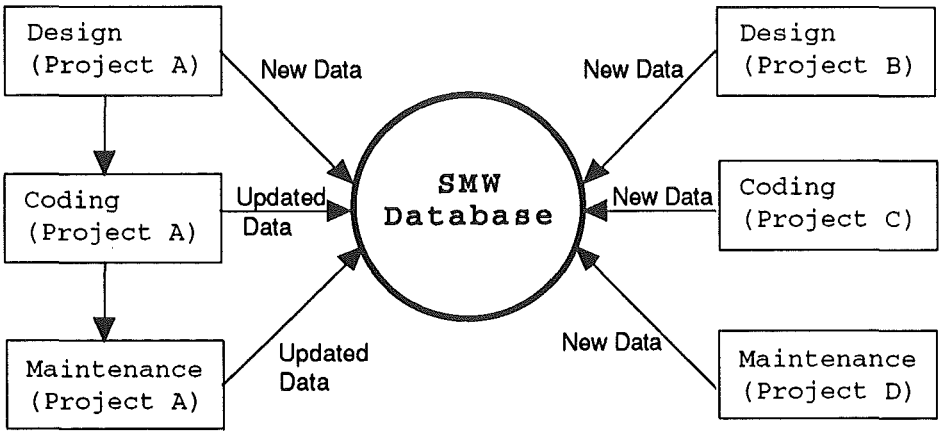


Figure 4.2 - SMW and different phases of development.

This leads to the second constraint of programming language independence. The SMW system should be able to store information from software products written in various languages of a similar nature. These could range from similar third generation imperative languages such as C and Pascal, but could also include other ways of describing software products such as the use of a generalized pseudocode or Nassi-Shneiderman diagrams[59] in the design phase.

[59]Tripp, L L; A Survey of Graphical Notations for Program Design - An Update; *ACM Sigsoft;* Vol. 13; No. 4; 39-44; 1988

As well as program entities common across the development cycle, program elements common between programming languages of a similar nature, in this case imperative languages. Again these entities will tend to be of a structural rather than a syntactical nature, for example a sub-program unit in C or Pascal. From the example about MAE described in Chapter 3, it can be seen that the addition of a new language for storage could involve designing new characteristic entities for the database for different syntactic elements or the same ones the behave differently (e.g. FOR statement in C or Pascal).

The third design constraint, that the database should be able to contain many different programs (Figure 4.2). And for each program in the database, the database should be able to store many versions of a program (Figure 4.3). This is because development at different stages is a process of examining alternatives[60], and often of repetition[61]. By being able to store many versions of a program, the history of a program can be kept, as well as a variety of different versions or design choices kept for comparative analysis. The maintenance of this historical data could be used in decision making based upon the complexity changes in earlier revisions on whether it would be better to rewrite a part of the software product rather than continue to maintain it.

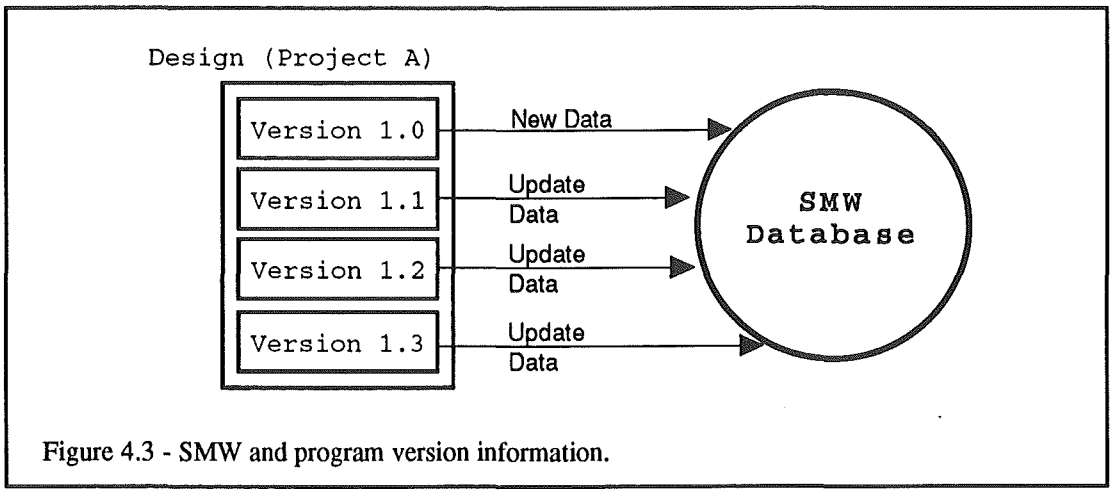


Figure 4.3 - SMW and program version information.

Also from a more practical point of view it makes more sense when faced with limited computing resources to store all the versions of a program in a single database, and indeed many different programs in the same database, rather than having the overhead of many databases being accessed and run all at once, and facilitates the collation of program data. It may be desirable in an environment with extremely limited resources to store only the differences between program versions in much the same way a revision control system might.

[60] Ince, D C and Hekmatpour, S; An Approach to Automated Software Design Based on Product metrics
[61] Reynolds, R G; The Partial Metrics System: A Tool to Support the Metrics Driven Design of Pseudocode Programs

The entities that were identified as being necessary for the storage of the program data desired are:

- Programs
- Program versions
- Modules
- Data Objects

Entities, Attributes and Relationships in the SMW Schema.

This section describes the entities in the database schema concerned with storage of program data that were identified when the SMW database was designed. Each entity's main attributes and relationships with other entities are discussed, with a fuller descriptions of the evolution of the schema described in the next chapter.

Programs and Program Versions.

The two types of entity, programs and program versions, need to be dealt with together. In the SMW system a program is made up of a collection of program versions. Each program version describes the program at a fixed point in time. This description of a program can be used to track the progress of the program through the development cycle, as well as being able to describe alternative designs and implementations that may occur, or need to be stored for comparison purposes. Figure 4.4 shows such a scheme with two alternative designs, and several implementations of each design needing to be stored.

The SMW system should also be able to take a sub-system that might represent an individual logical component in a program and treat this as a separate program.

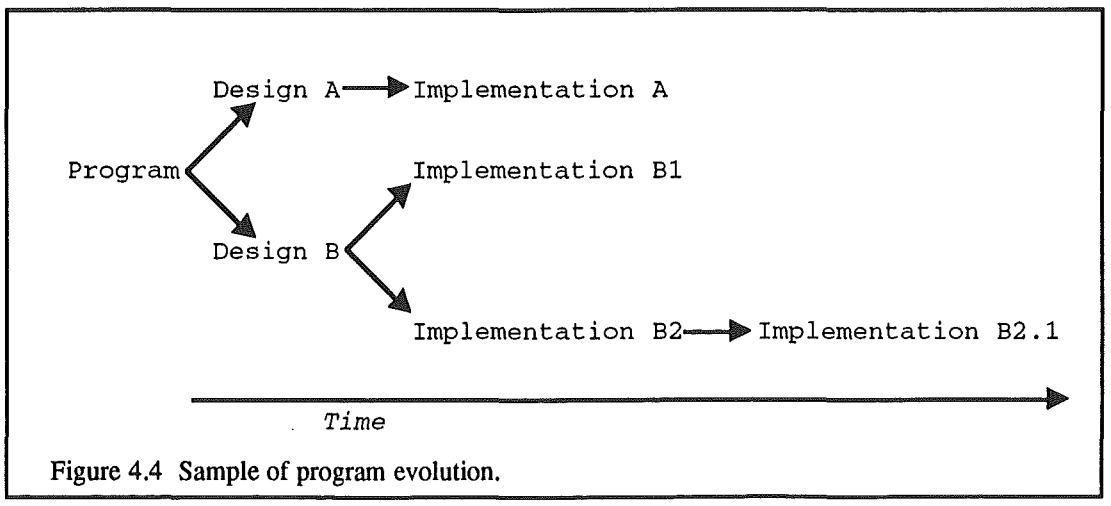


Figure 4.4 Sample of program evolution.

Thus, programs in the SMW system do not represent a single particular instance of a program but rather a set of instances of program versions, which in turn are collections of modules and data objects.

Modules.

Each program version is made up of at least one module. Modules in the SMW system represent the program unit, the sub-program. A function in C or a procedure in Pascal would correspond to a module in the SMW terminology. Modules are connected together by sub-program calls which invoke the program unit called and by storing the program calls between modules we can build up a picture of the program version's call graph which is useful for gathering structural information such as information flow between modules, cohesion and coupling.

Any value that is returned by a module to it's caller is useful in showing an information flow within the program. When a module does return a value the data type of the data value returned should be stored and the module noted as returning a data value.

Modules access data objects, such as variables and files, during their execution and relationships between the data object and module should be available in the database for fuller picture of inter-module relationships. It may also be possible to record the scope and lifetimes of data objects within the module.

The location of the module's definition within the software product that the program information came from is also useful for tools that might use it to open an editor at that place, or for studies on the partitioning of the design or module code between files. If the user of the SMW system doesn't have access to the files in the file system that belong to the software product then this may be the only way to access those files.

Data Objects.

The data object entity covers the range of objects that a module, or program version accessed to retrieve or store data of one sort or another.

The main class of data objects that needed to be captured in the database was the variable. The database should be able to store information on whether the variable in question is global or local, whether it is a parameter or argument passed from or to another module during a program unit call. By storing this data then information flows between modules that would not be shown by the program version's call graph can be analyzed. For example in Figure 4.5 the is an additional information flow between modules B and C that would only be captured by storing the information that modules B and C both access the global data object G.

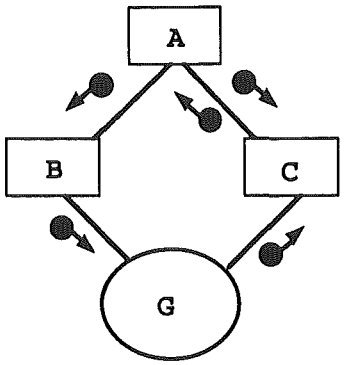


Figure 4.5 - Inter-module data flows.

Other not so obvious data objects that information that could also be collected are objects such as files, reports and external databases the program may generate or accesses. In the case of Function Point analysis being extended to the module level that even objects such as screens of data displayed or entered by a user could be included the data object class.

If complexity measures based upon information flow take the complexity of the data object itself into account, for example a field of a record being accessed might have a different value from the access of a simple integer variable, then it would be essential for the data type of the each object to be stored as well.

For determining coupling measures it would also be useful to store information on where a variable is accessed, altered (by having a new value inserted into it) or whether a module doesn't access the data object but passes it on untouched to another module, thus manifesting a type of stamp coupling.

As well as all these properties the location of the data object's definition (if applicable) may be useful information for any tools that would allow the user to look at the definition.

4.4.2 Metric Storage in the SMW Database.

As well as storing program data the SMW database must be able to serve as a repository or archive for any software metric data calculated for the program data stored. In order to do this the database must be able to store two different types of software metric data. These are:

- Information about specific software metrics.

- Any values calculated for a specific software metric.

In order for the SMW database to store any values calculated for a specific software metric there needs to be some information already in that database that describes the software metric.

For example, if a program was going to have it's size measured in lines of source code, then the database should contain the information detailing that the software metric "lines of code" exists and it's values can be uniquely identified. The name of the metric might also be stored, along with information that describes the location of the tool that generates the metric, the name of the tool and possible the type of software product that is analysed (e.g. C source code).

It should be possible for the above information to be stored in the database without any metric values being present. Such a case might exist if all the metric values for a metric for a program version have been deleted.

The metric values themselves are specific to a program version. Thus each metric value needs to be uniquely identified by the program version it's for, and as metric values can be calculated at several levels in a program version, for example function points for the program version as a whole, or cyclomatic complexity at a module level, then that levelling information should also be taken into account.

The SMW database should be able to store information on many metrics concurrently, as well a values calculated for each of those metrics for any program version stored in the database.

The metric values should be able to be accessed by the user or tools for analysis and re-use in further metric calculations, for example a size metric could be reused in the calculation of an information flow and then the new values created can be stored in the database. This storage of metric data prevents having to re-analyse the software product each time the values are required.

4.4.3 SMW Database Overview

Using the entities and constraints described earlier the following database schema was designed as the basic conceptual model for the SMW Database. Figure 4.6 shows a the schema in Entity-Relationship model form.

• Programs are made up of one or more program versions. A program cannot exist in the SMW system without an initial program version being stored also.

• A database can store more than one program.

• Program versions belong to only one program and a program version cannot exist without belonging to a program.

• Program versions are made up of at least one module.

• Modules can't exist without belonging to a program version. Module information is specific to that version. If two versions contain the same modules then there will be two separates module entries in the database, one belonging to each version.

• Modules may access one, many or no data objects.

• A module may call many other modules and in turn be called by many modules.

• A module may access one, many or no data objects.

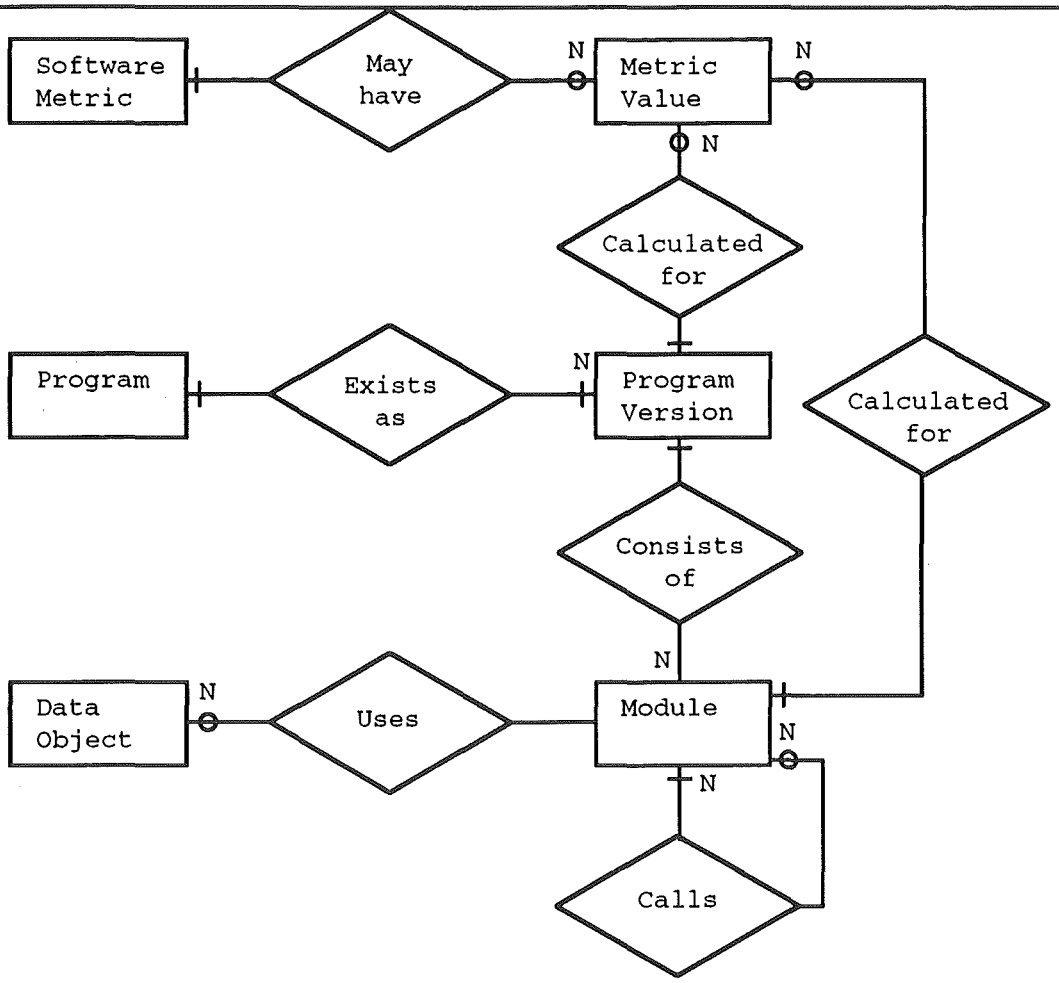


Figure 4.6 - SMW database design schema.

• Data objects may exist in a program version without necessarily being accessed by modules. For example, and unused global variable. This gives an extra relationship that may exist but isn't shown in Figure 4.6.

• Each version of a program may have metric values that have been calculated for each module in that version, and may also have metric values that have been calculated for the version as a whole.

• A software metric may exist in the system with no metric values calculated for it.

• A program version may exist in the system with no metric values calculated for it.

• Metrics may have been calculated at a program version, module or data object level.

4.5 The SMW Data Interface

The SMW data interface is the component of the SMW system that provides the necessary facilities for users or user processes, such as metrics analysis tools, to interact with the SMW database. Thus the SMW data interface was envisaged as the database management system for the SMW database working in much the same way as the process described below.

1. The user (or user process) issues an access request to the data interface, for example retrieving some stored metric values.

2. The data interface accepts request and analyses it.

3. The data interface then inspects the view of the database that is presented to the user, and maps that view onto the actual internal representation of the database.

4. The data interface executes the operation requested on the internal database structure.

5. If necessary the data interface will return the results of the access request to the user, mapping them from the internal structure back into a form compatible with the users view of the database.

The data interface is necessary because it provide the functionality described for standardizing the access methods to the database, allowing the sharing of data, maintaining the integrity of the data and providing levels of security,

The SMW data interface provides a standard "user interface" to the SMW database. This user interface is in reality an abstract view of the SMW database that describes the conceptual data model and the interface also provides a set of operations that the user can user to manipulate the data in the database at that abstract level. By limiting the operations possible on the database to those only provided by the data interface a tool that uses these operations always behaves the same way each time it is run. Also the operations allow manipulation at a conceptual level only meaning that if the underlying database structure is changed tools built using the data interfaces operations should always behave the same as before the change to the structure. Thus the interface provides a way of separating the SMW tools from the SMW database.

The data interface also aids in the sharing of data. There are two main ways of data can be shared, between users on the same database, and between users on different databases.

The first case is important because software development is often carried out by teams of people working on the same project. Thus information stored inside the SMW database should be accessible to more than one person. Also of the database is being used for archival and research

processed several users (or a single user) may be wanting to access different data sets inside the database as the same time for analysis.

The data interface should be able to control access to the database in such way as to allow several users or processes to access the database to manipulate the data with minimal effects on each other. This would allow the situation where users can be adding program information, retrieving metric values, and deleting data all at the same time with the data interface processing all requests in such a way that all the users can work concurrently and maintain a consistent view of the data that they are working with. Thus the data interface should be able to handle concurrent access as well as maintaining the integrity of the data. This is all carried out invisibly to the users. For example in Figure 4.2 there are several projects all be stored in the SMW database and it would not be unlikely that all of the might be accessed all at once.

The second case, data sharing between databases, is able to be provided by a data interface that is common to all SMW databases. Thus no matter how all the SMW databases are represented internally, conceptual level of the database is the same regardless of the hardware the SMW system is running on. Thus is the SMW data interface provides facilities for mapping information from the internal representation to the conceptual representation (or another intermediate form), and can map that information back again on another system then the data from one SMW database can be moved across to another SMW system running on another hardware platform.

The data interface should also be able to be provide the SMW database with some measure of security. Firstly by controlling what users can access the data in the SMW database and secondly by providing some method of information hiding. Information hiding can be supported by the creation of views of the data that are specific for individual users. For example, if a user wanted access to metric values for a particular project but the projects owners didn't want the name of the project, the module names or variable names known then a view onto the metric value data could be created which would provide the data values but with no other information apart from a generic program and program version indicator.

In the following chapter (Chapter 5) which describes the implementation of the SMW database the use of the commercial relational database management system Ingres is discussed as the means to providing the above data interface functionality.

4.6 The SMW Tools.

The third component in the SMW system are the SMW tools. These tools are used to used to a perform specific task or group of related tasks in the SMW system. Each tool exists as a separate component in the SMW system, separated from the SMW database by the SMW data interface which it uses to access the SMW database if necessary. Tools can be built for a range

of tasks and the SMW system is designed so that new tools should be able to be built quickly and easily, as well as old tools modified or discarded with little or no impact on other tools in the system. The main areas the SMW tools would perform their tasks in are:

- Program data collection.
- Metric data collection.
- Data retrieval, manipulation and presentation.
- SMW system administration.

4.6.1 Program Data Collection.

The program data collection tools need to be able to take a representation of the software product and use it to collect the program data that can be stored in the SMW database. For example, a particular tool might be able to take the source code or a design representation of a program and collect the information about the module call graph of the program, as well as the names and data types of the modules in the program. The tools should be able to recognize whether it is a new program being captured, or a version of a program already existing in the SMW database and act accordingly.

These tools and the tools that are described in the next section should be integrated into the development process in such a way as to minimize any disruption to the developers of the product. For example it may be possible to incorporate both the program data and metric collection tools into a revision control system in normal use, so that whenever a modified program is submitted to the revision control system the SMW database is updated accordingly.

4.6.2 Metric Data Collection.

Once the basic program data has been collected, then metric data can be calculated and collected for the particular programs stored in the SMW database. Once the metric values have been calculated by a tool then they are stored in the SMW database.

Metric tools can use a variety of methods to obtain the data to use for metric calculation. Some metrics tools might use the stored program information in the SMW database to calculate their metric values. An example of this would be to use the program information concerned with what modules call other modules to calculate the fanin and fanout values based on the program's call graph.

Other metric tools might use the metric data that has already been stored in the database to calculate the metric values, for example, using the fanin, fanout and lines-of-code values calculated earlier and stored in the database to calculate Henry and Kafura information flow

metric[62] values.

Metric tools might also directly analyze the representation of the software product to collect metric data and then store the results of the analysis in the SMW database. It would also be possible for tools to use data from all three sources, program and metric data in database and product representation in order to calculate the desired metric values.

These metric collection tools can work by using data that already exists in the SMW database (program, metric), by re-analyzing the software product outside of the database, or by a mixture of the above. Once the metric data has been calculated then the data can be stored away in the database for use at a later date.

4.6.3 Data Retrieval and Manipulation.

Once there is data in the SMW database, whether it be program or metric data there should be tools that allow that data to be manipulated and retrieved by the user. Once the data has been retrieved by the user there should be tools to present that data in a useful form.

An example of data manipulation and retrieval might be to retrieve the average metric values for all metrics calculated for a particular program version. This information could then be presented as a report, saved and then printed out.

Another example of a data retrieval tool is a tool that allows the user to interactively browse the contents of the SMW database, seeing what programs and program versions are in it, what modules a module calls or is called by, and what metrics the SMW system can store values for.

Another tool might retrieve data from the database based upon the user's requirements present that information as a graph, or in a form the could be used by another external tool. For example, a tool that plots two sets of metric values against each other based on the program version and metrics specified by the user, or a tool that retrieves the metric data required to an ASCII text file, that at a later date might be imported into a statistical analysis package for further analysis.

4.6.4 SMW System Administration.

Tools also need to be created to allow the user or users responsible for a SMW system to administer that system. These tools would allow a user to create a new SMW database or destroy an existing one, as well as tools to move data from one SMW database to another.

The addition of software metric information, such as the name of metric, to the SMW system so that metric values may be added to the system, is another example of a system administration

[62]Henry, S and Kafura, D; Software Structure Metrics Based on Information Flow; *IEEE Trans. Software Engineering;* Vol. SE-7; No. 5; 510-518; 1981

tool, as is a tool to remove information from the SMW database that is no longer required.

Chapter 5. The Implementation of the SMW Database and Data Interface.

5.1 Introduction

This chapter discusses of the implementation of the SMW database. Firstly a discussion of the evolution of the SMW database is given showing the way in which the database was structured changed as the database was modified and extended to bring it to a point where it could support the design goals. The second section contains a discussion of the tools used to implement the SMW database, namely the Ingres relational database management system, and how that system provided for the support of the SMW data interface.

5.2 Evolution of the SMW Database Implementation.

Several versions of the SMW database were implemented during the course of the project as design goals were modified, new goals defined and resource allocations changed. Full descriptions including names, sizes and data types of the database objects from all the versions implemented can be found in Appendix A. In this section the schema of each of the versions implemented will be discussed in brief with attention being drawn to aspects of each version and differences between versions. The versions of the database were names A, B, and C in respective chronological order with version A being the earliest version.

5.2.1 Version A.

Version A of the SMW database (Figure 5.1) was implemented at a time when the primary purpose of the SMW database was to capture the relationships between modules in a program. At this point in time the metrics storage facilities in the database were still being designed. Therefore the components of the SMW database that were implemented were the those concerned with the storage of information about what modules called what other modules, what global data structures a module accessed, and what parameter variables a module might be passed or return. The data type that of the value the module might return to it's caller was also captured. Thus the program structure at a module level, along with information flows within that structure could be captured.

In this version of the SMW database the components of programs being stored were identified by a unique program name and a module name that was unique for a program name. This allowed multiple programs to be stored in the database. If multiple versions of a program were to be stored then the program name would have to contain information to distinguish between them. For example the two versions of program "A" might have the program names "A.1" and "A.2" to identify them. This meant that in this version of the database that the relationship between versions of the same program was the same as that

between different programs.

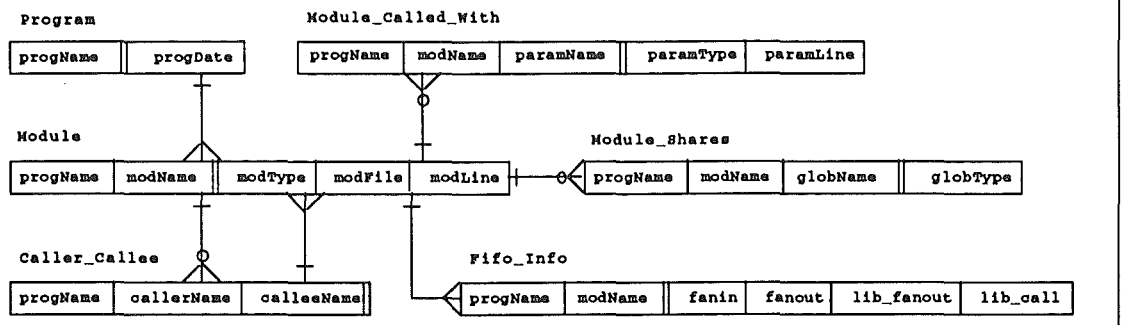


Figure 5.1 Version A of the SMW Database Implementation Schema.

The date the program structure information was captured by the database was stored, along with the program name in the "Program" table. Module information such as the name of a module, the data type it returned (if any), and the location of it's definition in the program (i.e. the line in a file the definition started) were stored in the "module" table.

Of the inter-module relationships to be stored there were three types: Modules calling modules, modules sharing global data and parameters passed to a module.

The first relationship was where a module called another module, or was called by a module. This was represented by the "Caller_Callee" table which stored, for each program, what modules called what other modules. The module doing the calling was known as the "caller" and the module being called was known as the "callee", and each distinct module call was stored. Thus this table allowed the cases of many modules calling a single module, and of a single module calling many modules to be stored simply.

The "Module_Shares" table stored the information on what global variables a module accessed. Thus to get all the modules who accessed a certain global variable then given the global's name and the program name all the modules with both of those attribute values could be retrieved. The table also stored the data type of the variable.

Parameters used by a module were stored in the "Module_Called_With" table. Again in a similar fashion the storage of global variable information, the information pertaining as to what program and module they belonged to was stored along with the parameter's name, data type, and line of the parameters definition in the file that the module was defined in.

Primitive metric information was also stored within this version of the database. The structural fanin[63] and fanout[64] values for each module were able to be stored for each

[63]Structural fanin: The number of modules that call a module.

[64]Structural fanout: The number of modules that are called by a module.

module, along with the structural fanout to library modules. In the SMW system a library module is a module that is referred to in a program, or software system but has no full definition in the software product being measured. For example, in a C program the function "printf" is part of the standard input/output library (stdio). Programs that use the printf function may contain information on the declaration of the function, such as the date type of the value it returns, and possibly even the number and name of the parameters is passed, but any information such as what global variables it may access or what other functions it calls is not available. Thus in the SMW database a module that is treated as a library module has no fanout value able to be calculated for it, has no global data stored for it, may have parameter data, and has a flag "lib_call" set if it is a module. All this information was stored in the "Fifo_Info" table in this version of the database.

5.2.2 Version B.

The next implementation of the SMW database, version B, introduced the concept of unique integer identifiers to identify program versions within the database, and modules within program versions. The integer identifiers were originally introduced for performance reasons but also had the effect of facilitating the storage of versions of programs.

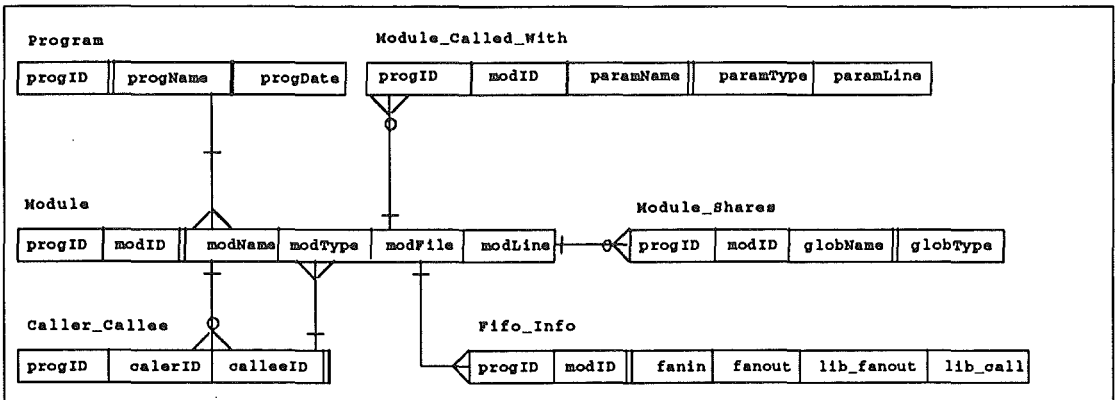


Figure 5.2 Version B of the SMW Database Implementation Schema.

The changes to the database that related to an improvement in the speed of the SMW database are related to the way that Ingres stores data. In Ingres, each of the tables in a database are stored as separate files. Each files is in turn divided into pages, and Ingres accesses table data one page at a time. The pages in the version of Ingres being used were two kilobytes in size, with forty bytes reserved per page for Ingres' overhead considerations. Each page is divided into records, and records may not span pages. The record length is the width of the row of the table with more overhead being incurred depending on the storage structure of the table, for example whether it is structured in a indexed sequential (ISAM) form or as a heap. To retrieve an entire table requires as many disk accesses as there are pages in a table, thus if more records in a table can be fitted in a page, then less disk accesses are required.

The use of integer identifiers meant that rather than repeatedly storing text fields for the program and module names, the text fields could be stored once, and an unique integer value for that text value stored instead. Changes to the contents of these text fields, such as a program name change, could then be applied once to the text in the location where the text name-identifier relationship was defined,rather than to all the places where it would exisit if the identifier scheme wasn't in use. These integer identifiers were determined at the time of the initial program information being entered into the SMW database.

By making the width of the records smaller, more records could fit per page in the database file. This improved performance by reducing disk accesses on the disk system where the database was stored, and if the database server was being accessed from a remote client machine then more data could passed though the network between them in the same amount of time. Table 5.1 is a table of the changes in the record length for each of the tables in versions A and B of the databases. The number of records able to be stored by page is also shown and these were calculated as:

$$\frac{(\text{page size - page overhead})_{\text{bytes}}}{\text{record length}_{\text{bytes}}} = \frac{(2048 - 40)_{\text{bytes}}}{\text{record length}_{\text{bytes}}} = \frac{2008_{\text{bytes}}}{\text{record length}_{\text{bytes}}}$$

In all the tables but one there was a sizable decrease in the record length and a corresponding increase in the number of records per page. The drawback to this system is the extra step to map the identifiers back onto text names whenever the test names are required.

Table	Record length (bytes)		Bytes saved per record	% saved per record	Records per page	
	A	B			A	B
Program	75	79	-4	-5.3%	27	25
Module	147	105	42	28.6%	14	19
Caller_Callee	166	12	154	92.8%	12	167
Module_Called_With	160	85	75	46.9%	13	24
Module_Shares	156	81	75	48.1%	13	25
Fifo_info	99	24	75	75.8%	20	84

Table 5.1 Storage space requirements for versions A and B of the SMW database.

The use of integer identifiers also allowed information to be stored that linked program versions together. By using the program name, all the program version identifiers that had that corresponding name in the "program" table could be retrieved. The program identifiers ("progIDs") are unique within the whole SMW database across all programs. Thus if a program version had an identifier of '5' the no other program version of any other program in the database will have this identifier and to get the name of the program the version belongs to the name that "progID" must be looked up in the "program" table.

5.2.3 Version C.

Version C of the SMW database (Figure 5.3) is the current version in use. It differs from it's predecessor, version B, in that it has explicit generic storage for software metrics, values that have been calculated for particular metrics and the level in the structure chart that the module can occur at (the "Module_Level" table). It also expanded the different types of data object that might have information stored about them in the SMW database, and the concept of type and location identifiers for data objects and modules was incorporated into the schema. These extensions involved the creation of seven new tables ("Metric", "Metric_Value", "Data_Object", "Data_Object_Def", "Type", "Location" and "Module_Level") and the removal of three of the tables in version B.

The table "Metric" contains information that describes a particular metric, in this case the name of the metric, an integer identifier that uniquely identifies that metric across the whole database, and the name of the program that generates the metric data. Thus the table might look contain records like the following:

```
<metID=1><metName="McCabe's Cyclomatic Complexity"> <metProg="mccabe">
<metID=2><metName="Structural Fanout"><metProg="fifo">
```

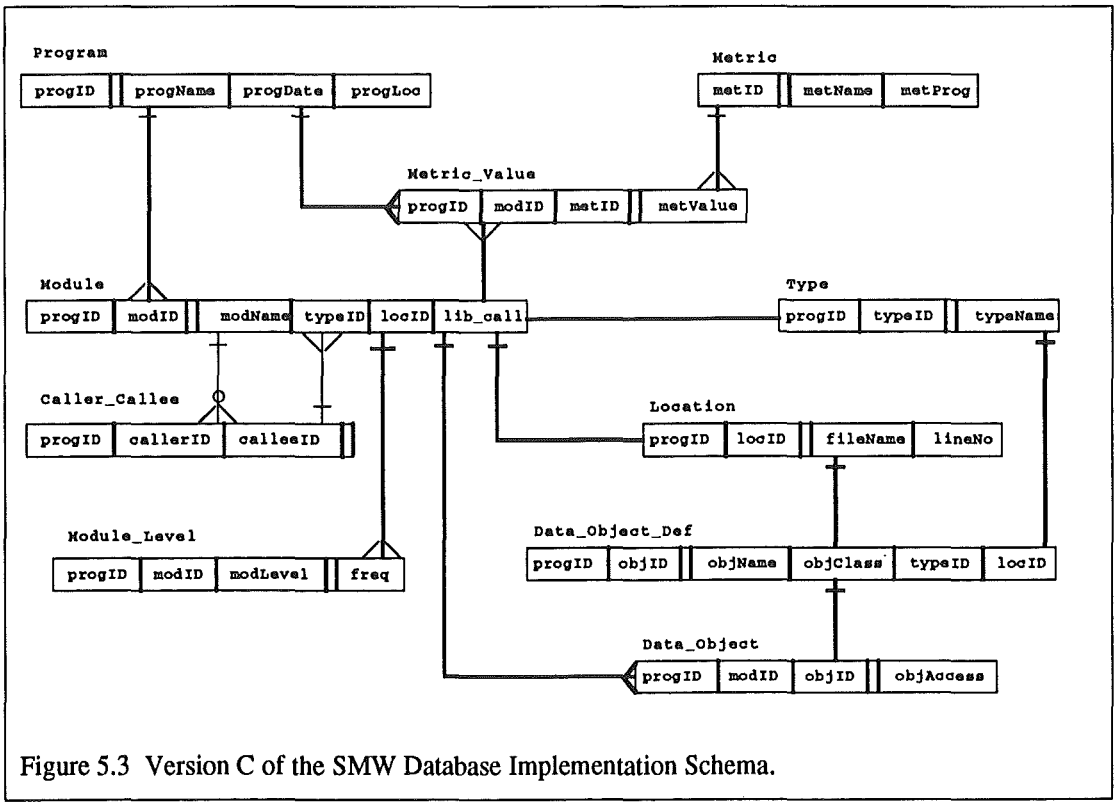


Figure 5.3 Version C of the SMW Database Implementation Schema.

The other table concerned with metric information is the "Metric_Value" table. This table is used to hold all the values that have been calculated for a metric that has been added to the system. A metric needs to be added to the "Metrics" table first and a "metID" assigned to it

before the metric values are calculated. The actual metric values themselves are stored as floating point numbers (taking up eight bytes of storage) as this proved the most flexible way of storing values that might be both floating point and integer. Values are stored in such a way that they are assigned to program modules in that a metric value is stored for a particular module in a program version (identified by "progID' and "modID"), and by the metric it's is for ("metID"). If a value has been calculated for a program version as a whole the module identifier '0' is assigned to it, along with the program version and metric identifiers. The addition of this table and the table "Metric" meant that the fanin-fanout information stored previously in the "Fifo_Info" table could be stored in the two new tables, and the "Fifo_Info" table was removed from the database. The library call flag the had previously been in the "Fifo_Info" table was moved into the "Module" table.

This version added two new tables concerned with the storage of data of data objects. Where as the previous versions of the SMW database were limited to storing only the global variables and module parameters accessed in a program version, this version added the concept of a generic data object that could capture information about data objects at an intra-module level as well as at an inter-module level. To do this the "Module_Shares" and "Module_Called_With" tables were replaced by the "Data_Object" and "Data_Object_Def" tables.

Each data object in a program version is assigned a unique object identifier ("objID") that identifies the data object within that program version. The information on what data objects are accessed in what modules for each program version is stored in the "Data_Object" table. As well as that the "Data_Object" table contains information on how the data object is used in the module. This information, stored in the "objAccess" field, describes whether the module is written to, read from, read from and written to, or not accessed at all. The use of an object identifier allows several data objects of the same name to occur within the same module. This is useful because it allows distinction between data objects of different scope in a module that all have the same name. For example, the variable "i" in Figure 5.4, which occurs twice at different levels of scope in function "A".

```
int A()
{
int i;
...
while (not_done) {
  int i;
  ...
  };
...
}
```

Figure 5.4 Function with variables of same name in scope in different places.

The "Data_Object_Def" table stores information on the definition of data objects in the SMW system. This information includes the name of a data object, it's class, that is whether it is a local variable, global variable, parameter, a file or another sort of data object, and pointers to the data type of the object and location of the data object's definition. The type and location pointers, "typeID" and "locID" respectively, are integer identifiers that correspond to information in two of the other new tables, the "Type" and "Location" tables.

The "Type" table stores information on the data types that occur in a program version. These data types might be used for data objects, or for the return values from modules. Therefore the data type information was removed from the "Module" relation and the pointer to a data type in the "Type" table inserted instead. This allows information for each data type to be stored only once in the "Type" table, rather than in each module and data object definition record.

The "Location" table works in the same way as the "Type" table with a location identifier pointing to a record in the location table that describes a location in the program version. The location table stores a file name and line number for a location, and the location of the file is specified by the value of "progLoc" in the "Program" table which contains the name of the directory the files exist in.

The use of integer identifiers to represent types allows the record widths of the module and data object definition tables to be narrowed again in the same manner that the program and module identifiers did in version B, thus allowing more records per database page for those tables.

5.3 Overview of the Ingres Database Management System.

One of the aims of the SMW system was to investigate the use of a database management system to support the collection and analysis of metric data in such a way as to maintain an archive of metric data.

5.3.1 Hardware, Operating System and Database Platforms for SMW.

The SMW Database and Data Interface were implemented using the Ingres Relational Database Management System[65,66]. This is a commercial product produced by Ingres Inc. (formerly Relational Technology Inc.) based upon a research project based at the University of California at Berkeley[67]. Ingres runs on a variety of operating system platforms such as DEC VMS, MS-DOS and various flavours of the UNIX operating system. Ingres version 6

---

[65]Ingres Inc. (formerly Relational Technology Inc.), *1080 Marina Village Parkway, Alameda, CA 94501*, U.S.A.

[66]Date, C J; A Guide to Ingres; *Addison-Wesley Publishing Company*; 1987

was used for the duration of the project.

The hardware platforms for the implementation of the SMW system were Sun Microsystems based. The bulk of the project was implemented on a Sun 3/60 workstation, based on the Motorola MC68020 C.P.U, with 8 megabytes of memory, and running version 4 for the SunOS UNIX operating system. Later in the project, the SMW system was moved to a Sun 4 SPARCStation, based on Sun's SPARC reduced instruction set architecture, with 16 megabytes of memory. The latter machine also ran SunOS version 4. These hardware platforms were multi-user with several users logged in to each machine, as well as each machine acting as an N.F.S. fileserver to other client workstations on the same network. The fact that the host computers were heavily loaded and that several users might be using different Ingres databases not associated with SMW at the same time as SMW was being used was the main reason that the changes to the database schema was made towards reducing the record widths of the database tables. As well there was also heavy use of the host machine for the compilation of programs, which might be typical of the evironment the SMW would have to operate in if used in a software development project.

5.3.2 Ingres as the SMW Data Interface.

This section discusses using Ingres to be the part of the SMW system known as the data interface. In Chapter 4 it was stated that the SMW data interface would behave in a similar manner to a DBMS in providing services to users and user processes, and that the main types of services to be provided were:

- Standard access methods to the SMW database.
- The sharing of data between users and databases.
- Security for the data in the SMW database.
- A basic "user interface" to the database.

Data Access

Ingres provides standard access methods to the SMW database though a variety of facilities. These features allow the user to manipulate and browse the database in a consistent way and by restricting access to the database through allowing these methods only the DBMS maintains the database in a integral state. The main access method for accessing the database is to use one of the two query languages that are provided with Ingres, that allow high level manipulation of the data. The use of these query languages is either through a monitor utility

---

[67] Stonebraker, M R, Wong, E, Kreps, P and Held, G D; The Design and Implementation of INGRES; *ACM TODS*; Vol. 1; No. 3; 1976

(described later in this chapter) or by embedding the query language inside another high level language program (described in Chapter 6).

The first query language that Ingres supports is QUEL ("QUEry Language")[68] which is based upon relational calculus. QUEL is the primary query language in Ingres and has been part of the Ingres DBMS since it's inception. Indeed many of the utility scripts that are generated by and used to maintain Ingres are in QUEL.

The second query language available is SQL ("Structured Query Language") developed by I.B.M.[69] which has been adopted as a standard for a variety of vendors DBMS's and has been the subject of an A.N.S.I. standardization committee[70]. SQL differs from QUEL in that it is based upon relational algebra.

Both query languages look similar (see Table 5.2 for a list of synonyms) and support both data manipulation (DML) operations such as adding, retrieving, updating and deleting data as well as data definition (DDL) operations such as creating and destroying tables, views and indexes.

QUEL	SQL
RETRIEVE	SELECT
REPLACE	UPDATE
DELETE	DELETE
APPEND	INSERT

Table 5.2 Corresponding Data Manipulation Commands in QUEL and SQL.

Through the use of these query languages and other tools such as forms and report tools (see later in this chapter) Ingres can control the access to the database, and the user can work at a higher level of abstraction.

Data Sharing.

Ingres provides the two data sharing capabilities that were desired in the SMW data interface. These were the sharing of data in a single database between more than one user or user process, and the ability to move data from one SMW database to another SMW database.

---

[68]Date, C J; An Introduction to Database Systems; *Addison-Wesley Publishing Company*; Vol. 1; 4th Ed.; 209-232; 1986

[69]Chamberlin, D D and Boyce, R F; "SEQUEL: A Structured English Query Language"; Proc 1974 ACM SIGMOD Workshop on Data Description, Access and Control; 1974

[70]X3H2 (American National Standards Database Committee). American National Standards Database Language SQL: Working Draft. Document X3H2-85-1 (December 1984).

(1) Sharing Data among users.

Ingres supports multiple user access to a database, and maintains data integrity during current operations by users. Thus, provided all the users are using the standard access methods, such as a supported query language, all the users should see the database as if they were the only users using it, and thereby be able to access and manipulate information in the database at the same time as another user is using the same information.

(2) Data Sharing between databases.

Ingres has two utilities to aid in the moving of data between databases. These are "copydb" and "unloaddb". Each of these utilities provide the ability to export the contents of a database to flat files in the UNIX file system, and to import the data back into another database. The utilities when run produce QUEL scripts that when run either export the data or import the data. The import scripts contain the data definition statements requires to build all the tables and indexes in the database, before the data is loaded into them.

The utility "unloaddb" also has the ability to store the exported data in ASCII text format, rather than the binary format used by "copydb". This makes moving the data in a database from one hardware platform to another less painful as no binary translation of the data has to occur to format it correctly for the new C.P.U. architecture. This facility of "unloaddb" was used for moving the contents of the SMW databases originally set up on the Sun 3/60 system over to the Sun 4 system, and worked without problem.

Another possibility with the storage of the data in ASCII text form is for it to be loaded into another database management system if this is desired, although all the internal database structures would have to be pre-created as the QUEL scripts probably would not work with those systems.

If using a database management system that supports the connection of different databases, such as Ingres/Star, then direct transfer of data between databases across a network is possible.

Security of Data.

Security of the data or controlling user access in the SMW database can be provided in two ways by the Ingres DBMS.

The most obvious way it can control access to the database is to only allow certain users to connect to a database. This is fine if you have only a few users but if the SMW system is going to be used by many different people using different sorts of tools then some other form of access restriction is required. Under SQL it is possible to grant privileges, such as

selecting and deleting information, on tables and even columns of tables to certain users. Thus it is possible for there to be tables that cannot be seen by some users, tables that can be seen by others but not updates, and tables that can be selected from, and updated by other users. Thus it is possible to make only the metrics tables readable by users who need metric data but don't need to know names of modules or programs, or you can have tables that are added by automated tools that don't have rights to delete data from those tables.

The second way that Ingres can provide data security is to have views defined of the base tables in the database. A view is a "virtual" table in that it appears to all intents and purposes to be a table to the user, and indeed behaves like one, but in actual fact it has no existence in it's own right and is derived from one or more underlying tables.

For example, Figure 5.5 shows a section of the "Module" table with some of the data stored in it. If the data identified by program ID '1' is sensitive then a view could be defined to hide that data. Also the user may not be wanted to know the location of the files so that information should not be supplied.

Module

progID	modID	modName	TypeID	LocID	lib_call
1	1	main	1	1	0
1	2	get_IRD	1	1	0
2	3	printf	0		1
3	44	read	0		1
4	7	process_graph	1	1	0
5	1	main	1	1	0
2	2	getname	2	1	0
1	3	get_tax	3	1	0

Figure 5.5 Module table showing a sample of data contained within it.

The SQL statement below creates a view "Module_View" that the user can look at with the sensitive and non-required data removed.

```
CREATE VIEW Module_View
    AS SELECT progID, modID, modName, typeID, lib_call
       FROM Module
       WHERE progID != 1 ;
```

This creates the view (Figure 5.6) and the user can access it like a normal table.

Module_View

progID	modID	modName	TypeID	lib_call
2	3	printf	0	1
3	44	read	0	1
4	7	process_graph	1	0
5	1	main	1	0
2	2	getname	2	0

Figure 5.6 The view "Module_View" showing data available for access.

Figure 5.7 shows the underlying base table with the data not in the view greyed out.

Module

progID	modID	modName	TypeID	LocID	lib_call
1	1	main	1	1	0
1	2	get_IRD	1	1	0
2	3	printf	0		1
3	44	read	0		1
4	7	process_graph	1	1	0
5	1	main	1	1	0
2	2	getname	2	1	0
1	3	get_tax	1	1	0

Figure 5.7 Underlying base table for view "Module_View" with data not in view greyed out.

Views have the drawback that they must be set up in advance by a user or administrator before the user can see them, but if views are the only thing the users know about then it may be that the privileges on the base tables never need to be set for individual users but rather only the views privileges.

Views could also be set up for basic statistical operations such as mean and standard deviation of a column of values, which could then be accessed by the user, rather than having to perform the calculations explicitly when the data was needed. These views would be automatically updated when new data was added to the base tables.

Basic "User Interface" to the Database.

Ingres also provides a set of tools that can be used to provide a basic user interface to the SMW database. These tools allow the user to interactively access the database to retrieve and manipulate data, to maintain the database tables and to generate reports.

Access to the query languages is possible through the use of the Ingres monitors. The Ingres monitors are interactive interfaces that allow queries specified in the appropriate query language to be typed directly in, run and then the results returned directly to the user. The results of the queries can then be browsed on the terminal screen and saved to a file if necessary. Ingres supports two terminal monitors, IQUEL (Interactive QUEL) and ISQL

(Interactive SQL), both of which are available to users of the SMW system.

The "Query-By-Forms" (QBF) tools allow users to issue simple queries against the database without having to have knowledge of a query language. It does this by presenting the user with a form on the terminal screen that displays fields that the user can enter data into for inputting into the database or using as search criteria. The fields displayed on the form can be for base tables in the database, such as "module" or "program", for views of tables, and for joins of several tables. The fields in the form also take the comparison operators , such as equals and less than, and the logical operators AND and OR.

```
Program Name: ProgA                          Date: 10/12/91

Module:

Metric: Cyclomatic Complexity

Metric Value: > 10




Help   Go   Query   End:
```

Figure 5.8 Sample QBF Form.

Figure 5.8 shows a sample screen form with search data entered in several of the fields to find all the modules in program "ProgA" captured on the date "10/12/91" that have cyclomatic complexity values of greater than 10. The menu options along the bottom of the screen can be selected by the user, and the form is a join of the "program", "module", "metric" and "metric_value" tables. These forms may be set up before hand by a user and saved and used by other users at a later date.

Ingres also provides tools for reporting. These are the report writer and the "Report-By-Forms" (RBF) tools. The report writer is run from the UNIX command line and given a table name or a QUEL RETRIEVE statement produces a report. For example to produce a report containing the names of all the modules in the program "ProgA" entered on the date "10/12/91" then the following view can be defined:

```
DEFINE VIEW MOD_NAMES ( MODULE_NAME = MODULE.MODULE_NAME )
       WHERE MODULE.PROG_ID = PROGRAM.PROGID
             AND PROGRAM.PROGNAME = "ProgA"
             AND PROGRAM.PROGDATE = "10/12/91"
```

and then following command can be executed and the report generated:

```
report mod_names
```

The RBF tool allows the format of the report to be designed as a form on the screen. This report definition can then be saved, and reused. The form definition gives greater control over the format of the report fields and allows the use of aggregate operators such as SUM and COUNT to be calculated for report fields.

Chapter 6. The Implementation of the SMW Tool Set.

6.1 Introduction.

The chapter gives an overview of the set of tools that were developed during the course of the SMW project. They include data collection tools for metric and program data, system administration tools for maintaining the SMW database, and a prototype user-interface, the SMW Browser. Detailed instructions on the used of these tools can be found in Appendix B, the user guide for the SMW system. This guide covers topics such as setting up the environment for the SMW system, a guide to all the screens and their functions in the SMW Browser, and formats of the SMW reports.

The development environment for the SMW tool set is discussed firstly, and then the individual tools themselves are discussed, with attention being drawn to issues of development, and facilities provided by the tools.

The tools developed were based upon the analysis of software products written in the C programming language[71]. The development environment contained a large quantity of program source code written in C that was freely available to a wide range of sites. Therefore the tools were developed with a mind to providing a test set of data that would be able to be obtained from many other sources, as well as testing the effectiveness of the SMW database to store source code information. The use of C source code would also provide a opportunity to compare a several code metrics, one of which had been specifically developed for analyzing C source code (NPATH[72]).

6.2 The Development Environment.

The SMW tool set was developed in the UNIX environment. The tools themselves were written with a range of development tools that were available. These tools include parser and lexical scanner creation tools, utilities and interfaces provided by the Ingres DBMS, a revision control system, a screen management library and a C language compiler.

The SMW tools were written mostly in the C programming language. The Sun Microsystems hardware platforms (Sun 3 and SPARCStation) running SunOS UNIX included a optimizing C compiler and linker that were used throughout the development process. A revision control system, RCS, was used to keep track of versions of the source code, maintaining backup copies of the files, as well a record of changes to each tool.

---

[71]Kernighan, B and Ritchie ,D; The C Programming Language; *Prentice-Hall, N.J*; 1978

[72]Nejmeh, B.A; NPATH: A Measure of Execution Path Complexity and it's Applications

The lexical scanner generator, Lex, and the parser generator, Yacc (Yet Another Compiler Compiler), were used to implement a scanner/parser combination that would process C source code, and provided access to the C grammar for the development of code metric tools.

The screen management library, Curses, which implements optimized terminal screen control, was used in the development of a user-friendly front-end to a program that captures the program data from C source files.

By far the most useful set of tools were those provided by the Ingres DBMS. Two main types of tools were provided. The first of these was the ability to embed query language statements within a C program. This made it easy to write tools that could be flexible and fast, while at the same time had the power and simplicity of the query language available for accessing the database. All the SMW tools that access the SMW database do so through the use of EQUEL (Embedded QUEL). This allows the program to connect to a database, which may be on a different machine to the one the tool is running on, and to insert, retrieve or delete data in the database at an abstract level. Thus the programmer doesn't have to understand the physical specifications of the database, or the communications protocols between machines, but rather can rely on the data interface, EQUEL, to handle all that for him or her. Figure 6.1 shows an example of EQUEL. Embedded SQL (ESQL) is also supported, as are a variety of host languages such as COBOL and FORTRAN.

```
## update_db (programID, moduleID, metricID)
## int programID;
## int moduleID;
## int metricID;
## {
## float the_value;

/* open the database "testdb" */
## ingres "testdb"

/* get the metric value for the specified program version,
   module and metric */
the_value = get_value(programID, moduleID, metricID);

/* update the metric_value table with the new value */
## replace metric_value ( metValue = the_value )
## where metric_value.progID = programID and
##       metric_value.modID = moduleID and
##       metric_value.metID = metricID

/* close the database */
## exit
## }
```

Figure 6.1 Example of embedded QUEL in C source code.

The other set of tools provided by the Ingres DBMS were the Forms and Menus tools. These tools, available from C, provided an easy way to implement screens that could be used to display data retrieved from the database. The menu tools allowed a series of menu and sub-menus of commands to be put together that were handled by the routines in the menu library automatically once the menus had been defined. The VIFRED (VIsual FoRms EDitor) allows forms to be developed interactively, and then saved into database for retrieval and update at a later date. Forms in that database could then be compiled to C source code and linked in with the tools, such as the SMW Browser, when the form was needed.

There was however an annoying problem using Lex and Yacc (or the GNU analogues Flex and Bison) with embedded QUEL. Getting Lex or Yacc to produce output with EQUEL actions, and preprocessing that output using the EQUEL preprocessor is straight forward, but after compiling the C source code produced, the program will fail in the link stage. This is because the Ingres libraries that are to be linked with the program's object code appears to have been created in part using Lex and Yacc and as such already have functions "yyparse()" and "yyerror()" defined. Thus the linker complains the functions "yyparse()" and "yyerror()" are multiply defined. As the libraries are unmodifiable, the solution to the problem has been to include actions in the makefiles of the programs that use Lex and Yacc, such that all occurrences in the C sources of "yyparse" and "yyerror" are replaced with another name. For example, in the program "mccabe" the function "yyerror" becomes "mc_yyerror".

Apart from this problem the Ingres tools provided a powerful and easy to use access to the SMW database from C programs, and the forms editor and menu tools allowed rapid prototyping of the user interfaces to the SMW database.

6.3 Tools That Were Implemented.

This section describes the tools that have been provided in the SMW tool set. It covers the following tools:

• Program Data Collection.

CFA - C-Flow Analyser used for program data collection.

• Metric Data Collection.

Fanin-Fanout - A metric collection tool to capture several inter-module metrics.

Proginfo - A metric collection tool to capture an intra-module size metric.

McCabe - A metric collection tool to capture an intra-module complexity metric (cyclomatic complexity).

NPATH - A metric collection tool to capture an intra-module complexity metric (NPATH).

• Prototype SMW User Interface.

SMW Browser - Tool to allow examination of SMW database and platform to launch other tools.

• SMW System Administration Tools.

SMW-programs - Reporting tool on programs in the SMW database.

SMW-metrics - Reporting tool on metrics in the SMW database.

Add-metric - Tool to add metric details to the SMW database.

Delete-program-metric -

Destroy-metric-info - Tools to remove metric information from the SMW database.

6.3.1 Program Data Collection - The C-Flow Analyser.

The C-Flow Analyser (CFA) provides the primary method of collecting program data from a collection of software product files for input into the SMW system. The information it captures is concerned with the flow or call graph of a program, and this information is stored in the relevant tables in the SMW database concerned with inter-module relationships of a caller-callee nature. It also provides information on the names of modules in the program, data types of values that modules return and preliminary call graph levelling information.

The CFA program is designed to be run before any of the metric data collection tools are used, as it defines the unique module, program and basic data type identifiers that the other tools use to refer to the program data by. CFA works by processing the output from a standard UNIX tool called "cflow", and works in such a way that is can be called easily from another program, such as a user interface screen, and can have the output from cflow redirected into it without the need for a temporary file. The "metsh" program described later is an example of a program that calls the CFA program.

Cflow is a UNIX tool available in System V releases of UNIX (and other systems with System V options) that analyses a collection of C, yacc, lex, assembler, and object files and then builds a textual representation of the flow graph of the external references, namely functions and global variables. It achieves this by in the following way:

(1) Files suffixed with ".y" (yacc) and ".l" (lex) are run through yacc and lex respectively to produce the C output files suffixed with a ".c"

(2) All C source files (".c" suffix) are the passed though the C preprocessor, cpp, and then run through the first pass of lint. This produces object files with a ".o" suffix.

(3) All assembler files are passed through the assembler, and the symbol table output from the assembler is used in conjunction with the object files by cflow to trace the graph of external references.

The sample output from the cflow program in Figure 6.2 can be seen in Figure 6.3.

```
1     main()
2     {
3       f();
4       g();
5       f();
6     }
7
8     f()
9     {
10      h();
11    }
```

Figure 6.2 Sample C program for cflow (line numbers shown only for illustration purposes).

As well as supplying the names of the functions, the data types where known are also shown, as are the locations of the definitions of the functions and the level in the call graph that the function is called at. The level of indentation in the output of cflow represents the level that a module occurs at in the call graph. It is this information the CFA program uses to construct the basic module, data type, location and caller-callee information about a program.

```
1     main: int(), <file.c 2>
2           f: int(), <file.c 9>
3                 h: <>
4           g: <>
```

Figure 6.3 Output from cflow after processing "file.c".

From the output of the cflow program in Figure 6.3, the following information can be obtained.

- Function "main" calls the functions "f" and "g".
- Functions "main" and "f" can return data values of data type "int".

- Function "f" calls function "h".
- Functions "main" and "f" are defined in "file.c" starting at lines 2 and 9 respectively.
- The data types of values returned by functions "g" and "h" is unknown.

The last case above, where the function's return data type is unknown, corresponds with the concept of a "library" call in the SMW database, which is a module that is called by a module but has no definition information available to it. Thus functions from the various libraries that are linked in with the object files by the linker to produce an executable program appear in this context. Before they are linked in their return value data type might be known by the programmer but the code definitions of those functions in the library is hidden. For example, the functions "g" and "h" above might well be functions such as "printf" and "fopen" from the "stdio" library.

The C-Flow Analyser program works in the following manner.

(1) Firstly the source files that contain the version of the program are run through the cflow program to produce a single output file that contains the flow graph of the program.

(2) The CFA program is given the name of the SMW database to input the data into, the name of the program and the location of the source files. It then assigns the program a new program identifier for the program in the specified SMW database and inserts a new entry in the program table for this program version.

(3) Then the CFA parses the cflow output file and writes records to a temporary flat file of the following form:

```
<function name,level in call graph,data type,source file,line number,caller name>
```

The function name is extracted directly from the call graph, as is the data type, source file and line number of the function definition the first time the function is encountered. Further occurrences of the same function result in the fields data type, source file and line number being left empty in the records concerning those occurrences.

The caller of the current function entry being processed is stored in the program in a stack. Whenever a function that occurs at a lower level that the current one being processed, the previous function processed is pushed onto the stack and the level count incremented. Whenever a function at the same level is encountered, the function on top of the stack is used as the caller. And when a function at a higher level than the previous function is encountered the top element of the stack is popped the number of times higher that the current function is than the previous one and the new top of the stack is the caller.

Thus the number of elements in the stack plus one represents the current level of the call graph being processed. For example, Figure 6.4 shows the progress of processing the call graph of the example in Figure 6.3. Before the function "main" is encountered, the level is zero and the stack empty. After processing "main" but before processing "f" the level in the graph is one, and after encountering the function "f" at a lower level than "main", "main" has been pushed onto the stack and the level incremented. Processing continues until the end of the call graph is reached. The remainder of Figure 6.X shows how the stack changes for each function processed.

Level equals stack height.

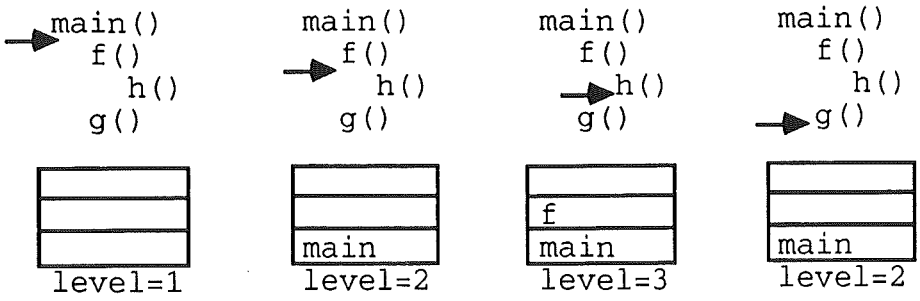


Figure 6.4 CFA use of stack to track module level in cflow output.

(4) The flat file is then read into a temporary table in the database using the QUEL "copy" command and the some of the SMW program information tables are populated for the program data being processed by using simple database queries on the temporary table. The file with all the records was copied into the database at the end, rather than inserting each record as it was created because it took less time this way.

The following information can be extracted from the temporary table.

- Data types.

CFA retrieves all the unique data type names from the table, assigns each type a unique type identifier, and inserts the type name, and program version and type identifiers into the "Type" table.

- Locations.

CFA retrieves all the unique file name, line number pairs from the temporary table and assigns each pair a unique location ID, and inserts the file name, line number, program version and location identifiers in the "Location" table.

• Modules.

CFA retrieves all the unique module (function) names from the table, and assigns each of them a unique module ID, and data type and location identifiers for the module's data type and definition location. If the module is a library module then the "lib_call" flag is set in the module record. The module name, library call flag and the program version, module, type and location identifiers are then inserted into the "Module" table.

• Caller-Callee Relationships.

All the unique module, caller pairs in the table are retrieved, and each name is mapped onto corresponding module identifiers. These identifiers and the program version identifier are inserted into the "Caller_Callee" table to record the inter-module calling relationships.

• Module Level.

The module name and call graph level information is retrieved along with the count of how many times a module occurs at a particular level. Module names are mapped to module identifiers and the level information is inserted into the "Module_Level" table.

(5) The temporary table is then destroyed and CFA terminates.

Using cflow as the source of data for the collection of program information has the drawback that the data it collects is reliable only for information about the flow graph of a program, not any useful information about data structures. Cflow has an option to include global data structure references in it's flow chart output but there is no differentiation between a global data structure and a function call. The use of cflow allows for the analysis of the coded software product held in a variety of forms, such as Lex and Yacc, allowing the creation of only one tool to gather the data, but the tool is limited to only those platforms that support cflow.

6.3.2 Metrics Collection Programs.

The SMW Tool set contains four different metrics collection programs that are used to calculate product metrics for program versions stored in the SMW database. Three of the programs are designed to work by analyzing C source code, and the other program uses information already stored in the SMW database to calculate it's values.

The programs collect inter-module metrics - fanin and fanout, a size metric - number of statements, and two control flow metrics - NPATH and McCabe's cyclomatic complexity. The last three tools are based upon a C scanner and parser combination.

All the metrics tools work similarly. Each program is given the name of the SMW database

to use, and the program identifier of the version of the program to calculate the metric values for. The program then connects to the database, retrieves the metric identifier for the metric it is going to calculate, and then processes it's input data, whether the data is from the database or and external source. The metrics collection tools are designed to be called from the SMW Browser program, and as such have no checks in them that the program identifier supplied is valid, or that the metric information hasn't already been supplied. These checks are left to the program that calls the metric collection tools, but the metrics collection tools can be run from the command line of a UNIX shell if the user is confident that the program version exists and the metric values don't yet exist. This information can be determined by using several of the SMW system administration tools described later.

Once metric value information has been calculated for each module then each of the tools stores that information in the "Metric_Values" table in the database.

6.3.2.1 The Fanin-Fanout Tool.

The "Fanin-Fanout" tool calculates three different values for each module in a program version. These are:

• Structural fanin - the number of modules that call a module.

• Structural fanout - the number of modules called by a module including calls to library modules.

• Structural fanout to library modules - the number of library modules called by a module.

This information is calculated from the information stored in the "Caller_Callee" table in the database, that maintains a records of what modules the modules in a program version call.

Thus for each module fanin can be calculated by:

1. Retrieving all the modules from the "Module" table for the specified program version

2. Counting the number of caller modules in the "Caller_Callee" table that call each of the modules retrieved from the "Module" table.

Fanout is much the same, with a count of all the modules that are called by the modules retrieved from the "Module" table instead. To calculate fanout to library modules, a temporary view of the "Caller_Caller" table is set up with only those callee modules in it that are specified as being library calls in the "Module" table. The view is then used instead of the whole "Caller_Callee" table, and the fanout values are calculated the same way as previously.

The "Fanin-Fanout" tool is a good example of the benefit of the storage of program data in the SMW database. Once then program data has been initially collected by the CFA program then the fanin and fanout metrics can be calculated with no further contact with the original software product required.

6.3.2.2 C Grammar Based Tools.

The three tools that compute the NPATH, McCabe and statement count metrics are based upon a C language lexical scanner and parser that directly analyses the products source files and computes values for each function in those files. Information about the NPATH metric may be found in Appendix C, and the definitions for the scanner and parser can be found in Appendix E.

The scanner[73] and parser[74] that were used for these tools are adapted versions that were available from a Public Domain source. The original C parser was a faithful implementation of the ANSI C grammar that described by Kernighan and Ritchie[75] and formalized by the ANSI technical committee X3J11[76]. The parser was written in a grammar description able to be processed by the Yacc tool to produce a C source file that was compiled into each of the tools. The scanner was described in such a way as to be processed by the Lex scanner generator, which like the Yacc program, produced a C source file containing a compilable scanner.

Certain modifications and extensions needed to be made to both the scanner and the parser due to limitations in the original definitions, the environment which they were being used in, and the nature of the different metrics. The first two types of modification are described next, with modifications for each metric discussed separately.

The first modification that needed to be made was to bring the names describing the tokens that the scanner would pass to the parser into line with those that the parser was expecting. The names were changed in the scanner, and the actual token values that the scanner passed to the parser were defined when the parser description was processed by Yacc, and then included in the source file of the scanner produced by Lex.

The second modification of that was made was to extend the scanner to recognize the tokens for floating point, exponential, hexadecimal, character and character string constants.

---

[73]Sanders, T. C lexer. Posted to comp.lang.c, USENET News, 1990

[74]Kumar, S. BNF ANSI C Parser. Posted to comp.lang.c, USENET News, 1990

[75]Kernighan, B and Ritchie ,D; The C Programming Language; *Prentice-Hall, N.J.*; 2nd Ed.; 1988

[76]Prosser, D F; American National Standard X3.159-1989, Programming Language C; *American National Standards Institute,New York, N.Y.*; 1989

Primitive rules for recognizing the character and string constants already existed but these didn't cover the cases when a string might be carried over several lines using the '\' escape, or cases where the character constant contained a single quote. An example of a constant spread over more than one line is the string constant used as the format string in the following statement:

```
printf ("hello\
world");
```

The next step was to modify both the parser and the scanner in order to "de-ANSI-fy" the tools. All of the source code that was currently available existed in pre-ANSI standard form, and the tools had to be adjusted to cope with this. The main changes were to function declarations, and to the reserved word set. ANSI C expects function declarations of the form:

```
int A_function ( char *parameter1, int parameter2)
{
/* function body */
...
}
```

In contrast, the pre-ANSI function definition could look like the following:

```
A_function ( parameter1, parameter2)
char * paramter1;
int parameter2;
{
/* function body */
...
}
```

The parser was changed to accept the older style of function definition as well as the later style, so that code in both styles can be analyzed. The other change was to remove those reserved words in the ANSI standard that didn't exist in the pre-ANSI language from the scanner, so that it they were encountered they would be treated as an identifier or such like instead.

The final major change to the scanner-parser combination was the one with ramifications for the collection of metric data. The modification that needed to be made occurred because the original scanner had no way of determining whether the token that it had was either an identifier or a typedef name. The same pattern matching rules in the scanner matched both of these types of token, and the author of the grammar had documented the fact that some form of differentiation between the two sorts would be necessary. For an example of typedef and identifier confusion the following declaration is given. Thus when the scanner comes across the token "Name" it would pass it as an identifier to the parser, which as it was expecting a

typedef name token would produce an error and terminate.

```
typedef struct my_struct {
     unsigned short  ID;
     char            *text;
} Name;
```

The method used to overcome this was to create a list of known typedef names that were currently in scope. The scanner checked all identifiers against this list, and if it found a match then it passed a typedef name token to the parser, otherwise an identifier token was passed. Whenever the depth of scope inside a function changed the typedef list was checked for out of date typedefs, which were then removed. This identification of typedef names involved isolating the parse structure for typedef declarations, and the parser accepting an identifier as the typedef name in the declaration. Once that identifier was obtained by the parser it was added to the typedef list, with it's scope level, for identification by the scanner at a later time. Problems can arise with typedefs and identifiers with the same names occurring but the current scheme seems to work well with the source code analyzed so far.

The final aside in this scheme of things is that for the typedef declarations to be included at all quite often required the C preprocessor to be run over the source code to be analyzed. This would include the typedef declarations from any included header files but had the side effect of expanding any macro definitions in the source files. Thus the metrics would measure the preprocessed code and any extra complexity that the programmer had gone to the trouble of hiding in a macro. This expansion can be seen as both good and bad, in that the code analyzed is not what the programmer "sees" but is instead the true source code with all the intervening facade of macros removed. As all the tools based on the C scanner and parser preprocess their input they are all working on under the same rules, rather than some tools doing one thing, and another something else. Thus at least a consistent base for measurement is achieved across all the tools.

• ProgInfo - A Module Size Metric Collector.

The proginfo (program information) tool performs the simplest metric analysis of the three metric analyzers based upon the C scanner/parser combination. It's purpose is to collect the size of each function in the C source file measured in the number of statements. That is, a count of all the statements that are responsible for the actions that are performed in a function, rather than the parts of the function that are responsible for defining the structure of the data objects within the function or the function interface. The metric is easily calculated from the parser definition of a statement that is show below:

```
Statement  :    Labeled_Statement
           |    Expression_Statement
           |    Compound_Statement
           |    Iteration_Statement
           |    Jump_Statement
           ;
```

When ever any of the statements other than the compound statement are processed at this point in the grammar the number of statements for the current function is incremented by one. Compound statements are left alone as they are made up of combinations of the other types of statement.

Given the program identifier and a database name the program inserts a new value into the "Metric_Value" table in the specified database for each module (function) it processed from the input source file.

• McCabe - A McCabe Cyclomatic Complexity Metric Collector.

The mccabe program calculates the cyclomatic complexity of each function in the input source file. It takes a destination database and a program identifier as arguments, and stores it's results in the "Metric_Value" table of the specified database.

The metric is calculated on as the number of decision statements in a function plus one, where multiple conditions in decision statements each count as a separate decision statement. For example, in Figure 6.5 the first IF statement counts as adding one to the cyclomatic complexity, but the second IF statement adds two.

```
if ( a == b)                    if ( a==b && a==d)
  c();                            c();

 adds 1                          adds 2
```

Figure 6.5 Decision Statements with total added to cyclomatic complexity.

Decision statements that were used to calculate the metric where "IF", "WHILE", "DO-WHILE", "SWITCH", and the "?" (conditional) operator. Upon encountering one of these the scanner set a flag that let the parser know that the number of expressions in the condition where to be added to the cyclomatic complexity value. At the end of the function this value had one added to it and was stored in the database for the module the function represented.

• NPATH - Another Intra-Module Flow Graph Metric Collector.

The npath tool collects the NPATH metric values for each of the functions in the input source file. The collection of these values is done in the parser component of the program, with the

terminal symbols in the grammar being assigned base semantic values. As the program parses a function, these values are manipulated according to the rules for calculating the NPATH metric, and become combined into the various semantic values for the non-terminal symbols. For example in the section of code below (Figure 6.6) the values returned from the processing of the non-terminal symbols "If_Condition" and "Statement" are used to calculate the NPATH value of an IF statement. The NPATH value of an IF statement is defined as:

NP(IF) = NP(<if-range>) + NP(<expr>) + 1

```
If_Statement : IF_TOKEN
               If_Condition
               Statement
               { $$ = $3 + $2 + 1; }
               ;
```

Figure 6.6 NPATH calculation for IF statement in C parser.

This method of calculating the NPATH values of statements in a C program worked well with one exception that required extra work to achieve. The SWITCH statement value is based partly upon the sum NPATH value of each of the CASE statements within the SWITCH statement. The standard grammar of CASE statements being a type of LABELED statement meant that it was hard to sum their total values, because there case no way to logically link successive CASE statements within a SWITCH statement. One could not sum the values of the statements that existed within the SWITCH statement's compound statement, as the compound statement was a generic symbol that was used outside the SWITCH statement as well. Thus adding values within the compound statement would affect the calculation of other NPATH values.

The problem was fixed by redefining the grammar for the SWITCH statement so that instead of using the generic compound statement, it used it's own compound statement "Switch_Body_Statement". This was made up of CASE and DEFAULT statement lists and allowed the special case to be catered for. Figure 6.7 and 6.8 show the differences between the two SWITCH statements.

```
Switch_Statement : SWITCH_TOKEN
                   Switch_Expression
                   Statement
                 ;
```

Figure 6.7 Original SWITCH statement.

```
Switch_Statement : SWITCH_TOKEN
                   Switch_Expression
                   Switch_Body_Statement
                 ;
```

Figure 6.8 Modfied SWITCH statement.

6.3.3 SMW Browser - A Prototype User Interface to SMW.

The SMW browser tool is a prototype user interface to the SMW system. It offers the user a range of facilities, available from a centralized menu hierarchy, that allow the user to access information in a SMW database and to execute other SMW tools easily.

The SMW browser was developed using the Ingres Forms and Menus tools, and presents the user with an interactive user interface that is controlled by menu options. Information is displayed to the user through various forms, and the user reacts to the information by choosing the appropriate menu options. Figure 6.9 shows the initial title screen that the user of the SMW browser is presented with upon starting up the program. The menu at the bottom of the screen shows the user what options are available and in this case the options are to access an existing SMW database, to create a new SMW database, to destroy an existing SMW database or to quit the program. The last option "Quit" is not shown in the figure but becomes visible upon pressing the "Menu" function key that shows other available menu options.

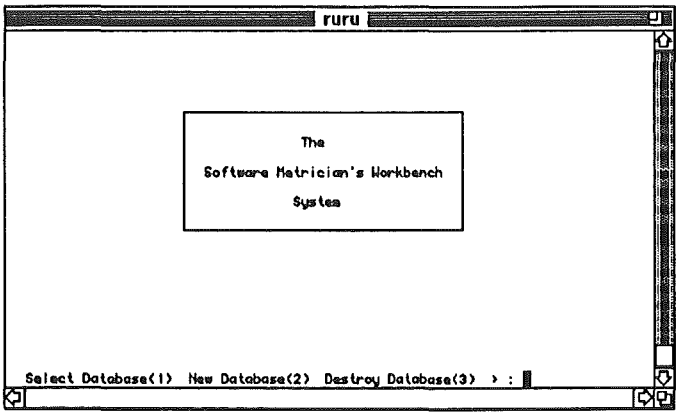


Figure 6.9 SMW Browser Title Screen.

The SMW browser provides access to the following facilities:

- Creation and destruction of SMW databases.
- Examination of the contents of an SMW database.
- Deletion of program and program version data from the SMW database.

• Generation of reports on program and metric information.

• Generation of plots for comparison of two metrics.

• Access to data collection tools, both program information and metric data.

• Access to the Ingres query sub-systems IQUEL, ISQL and QBF.

The SMW browser supports the creation and destruction of SMW databases. When the user chooses to create an SMW database from the browser, the browser will create a new Ingres database, with the name specified by the user, containing new SMW tables and their respective indexes. When the user chooses to destroy a database then the entire Ingres database specified, with all it's data, is destroyed by the browser.

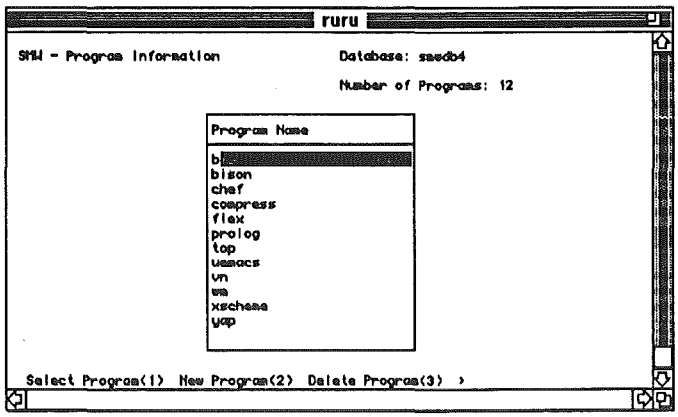


Figure 6.10 SMW Browser Program Information Screen.

The browser, as it's name suggests, also allows the user to examine the contents of the SMW database that was selected at the title screen. Figure 6.10 shows the screen that displays a list of all the programs that are stored in the SMW database. The user can move the cursor through the list of programs and then select one with the menu option "Select Program". This causes another screen to be shown all the versions of that program that exist (Figure 6.11). Selecting a version in a similar way to as on the previous screen causes another screen showing the inter-module structure of the selected version to be displayed (Figure 6.12). In this way the user can examine the contents of the database.

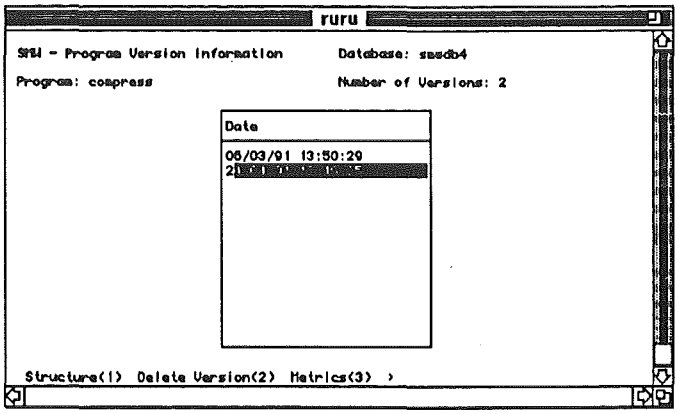


Figure 6.11 SMW Browser Program Version Information Screen.

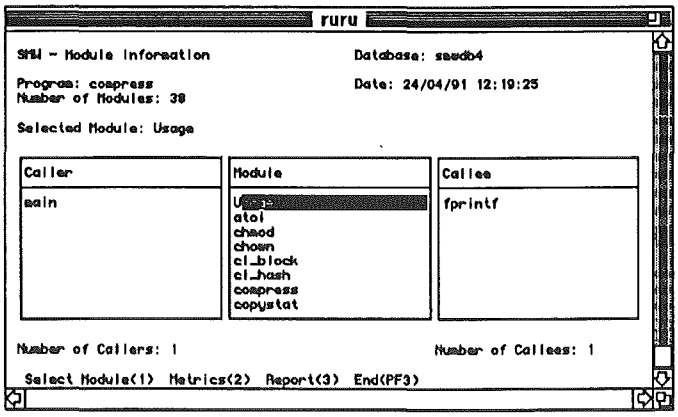


Figure 6.12 SMW Browser Module Information Screen.

Metrics that have been calculated for the program versions can be displayed. The SMW browser can provide a information of those metrics the SMW system can currently store and what metrics have been calculated for a version, as well as providing a query screen at program version level to allow metric values to be selected based on the name of the metric and a value condition. Figure 6.13 shows this query screen. The "Select Metric" menu option brings up a list of metric names that have been calculated for the version of the program, and the user chooses a metric from that list.

Using the browser the user can delete information about a program, or a program version. In the first case, if a program is deleted then all the information about all the versions of that program is deleted. If just a version is selected for deletion then only that version's information is destroyed. The information destroyed includes all the program information such as module, location, and data object information, as well as any metric information calculated for the specified program or program version.

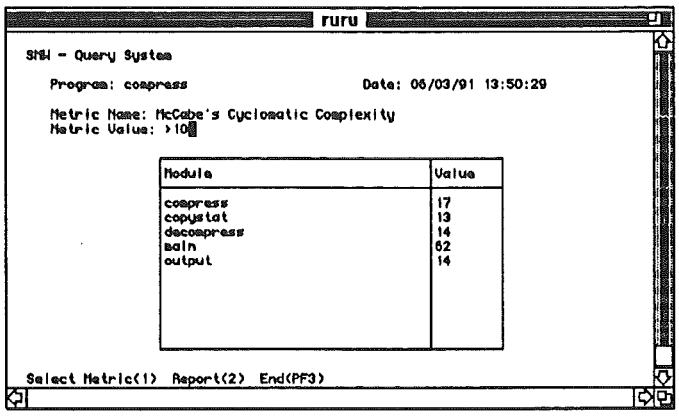


Figure 6.13 SMW Browser Query Screen.

At almost any point in the browser there is a reporting option the allows information on the screen to be save to a text file. Reports can be generated for the following topics.

- Programs in an SMW database.
- Programs in an SMW database, with information on the date of each program version.
- All the metric values for all the metrics calculated for the specified program version.
- All the metric values for a selected metric calculated for the specified program version.
- All the metric values retrieved from a query performed at the Query screen (Figure 6.13).
- All the modules in a program version, what modules they're called by and what modules they call.
- A module in a program version with what modules it's called by and what modules it calls.

These reports can the be imported into a statistics package or a spreadsheet for analysis, or can be printed out for a hard copy of the data.

The SMW browser program also interacts with other software tools in the UNIX environment. Some of these were specially created during the course of the project, where as other were utilities that were already available. An example of the latter is the plotting facility provided by the SMW system.

The browser allows the user to plot the value from two metrics against each other in a scatter plot format. It does this by accessing the GNUPLOT plotting package. GNUPLOT is a command-driven interactive function plotting program. After the user selects the metrics that are to be plotted against each other at the Plotting screen, the browser generates a data file

with the appropriate GNUPLOT commands in it with the metric data, and then invokes the GNUPLOT program with that data file. If the user is running on a terminal that can support Tektronix 4014 emulation the plot, with labelled axes and title, should be displayed. For example on an Apple Macintosh running the NCSA Telnet terminal emulator a window will appear with the plot on it. This window can the be printed. GNUPLOT can produce output in a variety of formats, such as the Adobe PostScript page description language, so by changing the output requirements plots can be stored as text files and previewed and printed at a later date. Appendix G gives an example GNUPlot output for the "compress" program from the test data set of programs.

Other custom built tools such can be added as well to produce desirable input files for other plotting packages.

The SMW browser also provides a mechanism for the collection of program information data, and metric data through the accessing of a program that uses the CFA program, and by calling the metric collection programs described earlier.

When the user is at the "Program Information" screen that displays the programs in the SMW database the user may choose to add a new program or program version. When this menu option is selected the user will be prompted to use either the "Metric" shell or the UNIX shell. In the latter case the browser initiates an interactive UNIX shell session and the program information can be gather by manually using the CFA and cflow programs. If the user selects the "Metric" shell the browser asks for the pathname of the directory containing the source code, and the name of the program to add. It then executes the "metsh" program run the "Metric" shell.

The "Metric" shell is a separate program that allows the user to select interactively the files to run through the cflow and CFA program. The program displays the contents of the specified directory on the screen, and allows the user to move a cursor onto the file names and then to view or select the files in the directory. If a directory is selected the user changes directory to that directory and the new directory's contents are displayed. If the user selects "Analyser" then all the selected files are run through the cflow program and the output of that is run through the CFA program. The user is then returned to the SMW browser. Figure 6.14 shows the "Metric" shell screen with a variety of selected files.

The SMW browser can also run the metric collection programs directly so that the user needn't to have to run them from the UNIX shell. At the "Program Version Information" screen the user can select the "Calculate Metric" option which will bring up a list of metrics that can be calculated.

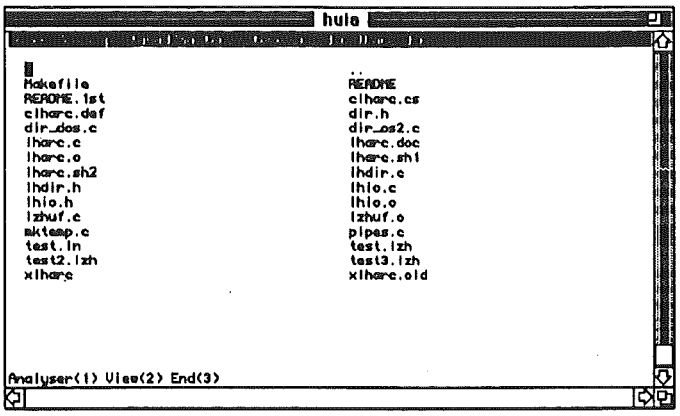


Figure 6.14 The Metric Shell.

If the user selects a metric that has not yet been calculated for this program version then the program gets the locations of the metric program to run from the "Metric" table file "metProg" and the program version source file directory pathname from the "Program" table file "progLoc", and then processes each of the version's source files in the "Location" table with the C preprocessor and metric program. When this is finished the browser notifies the user, and another metric can be calculated if necessary.

The Ingres query language monitors, IQUEL and ISQL, as well as the Query-by-Forms (QBF) sub-system are available from inside the SMW browser to give the user extra flexibility with information retrieval. These sub-systems are activated off the "Program Information" screen menu, and at there termination the user is returned to that screen.

6.3.4 System Administration Tools.

The SMW tool set includes several system administration tools. These tools are currently run from the command line and are used for two tasks. The first purpose is to provide two reports about the contents of a specified SMW database, and the second is to add and remove information about metrics stored in an SMW database.

There are two reporting programs that provide information about programs in a database, and metrics in a database respectively. These programs are designed to be able to be run without the overhead of the SMW browser, and can run unattended as they do not require a user to select menu options to generate a report. Thus they can be set up to run from a batch file or UNIX shell script. The two programs are "smw-programs" and "smw-metric".

The "smw-programs" program produces a report about each program version that exists in the specified database. This report includes the name of the program, it's date of capture, it program identifier, the location of it's source files, and the names of the source files. It can

be used by the user to obtain the program identifiers that other programs, such as the metric collection programs, require to run.

The "smw-metrics" program produces a report on each metrics that exists in the database, including the metric name, along with the name of the metric program that generates the metric and the metric identifier.

The other programs are concerned with the addition and deletion of metric data. The three programs that perform these tasks are "add-metric", "delete-program-metric", and "destroy-metric-info".

The "add-metric" program adds a new software metric to the SMW database specified. It adds the metric's name and metric program used to generate the metric values. It also generates a unique metric identifier for the new metric. This information must be added to the database before any metric values can be calculated for the new metric.

The "delete-program-metric" program deletes all the metric values for a specified metric in the database that belong to the specified program version. The program version is identified by the program identifier, and the metric by name.

The "destroy-metric-info" program destroys all information about the specified metric from the SMW database. It deletes all the metric values for the metric specified and the removes all the information from the "Metric" table concerned with the metric.

These last three programs could be integrated into the SMW browser tool at a later date to provide that tool with better all round system administration capabilities. It is envisaged though that the addition and deletion of metrics to an SMW database will be a fairly rare event, except maybe at the establishment of new SMW database, as compared to users adding and deleting program versions and metric values, and will probably be carried out by a specific administrator. Thus the tools are separate at present and able to be used in batch mode. For example, after a new database has been created the SMW administrator could run a batch script that adds all the desired metrics to the SMW database, rather than interactively specifying each one in turn.

6.3.5 Stand-alone Tools.

Tools for metric analysis have also been developed that can be run from the command line, and accept their input from a specified file. The output from these programs is written to the screen or can be captured in a file. No database interaction is required. Currently these tools exist for the mccabe, NPATH and number of statements metrics.

Chapter 7. Metric Analysis Case Studies Using the SMW System.

7.1 Introduction.

This chapter describes some uses for and analyses of software metric data that can be provided by the SMW system. A series of analyses are presented demonstrating how the data that SMW can provide could be used. These include using the metric data to obtain statistical information about the structure of a program, validating results of analyses published by other authors, and examination of relationships between metrics.

The data used for the analysis was provided by analyzing a set of twelve programs available in the public domain. These programs included two text editors (Chef, MicroEmacs), two programming language development tools (Flex, Bison), two programming language interpreters (CProlog, XScheme), and a variety of utilities. These utilities included a text file browser (Yap), a compression program (Compress), a bibliographic utility (Bib), a Usenet news reader (VN), an system performance monitor (Top), and a multiple window terminal emulator (WM). The tasks that the programs perform cover a wide range of activities and make sure that programs from a variety of application contexts are used (Table 7.1). Programs with source code readily available ware also chosen so that a test set of data could be analyzed that would be available to other researchers at different locations.

Program	Total Number of Modules	Application Type
Bib	93	Biblographic Utility
Bison	150	Parser Generator
Chef	227	Full Screen Text Editor
Compress	38	File Compression Utility
Flex	124	Lexical Scanner Generator
Prolog	161	Prolog Interpreter
Top	108	System Monitoring Utility
MicroEmacs (uEmacs)	447	Full Screen Text Editor
VN	176	Usenet News Reader
WM	140	Window Manager Utility
Xscheme	502	Scheme Interpreter
Yap	144	Text File Browser

Table 7.1 Information on the programs in the test data set.

The test set of programs provided 2310 modules to be stored in the SMW system. Of those modules all of them had fanin metric values calculated, and a subset of 1825 were available to have fanout, cyclomatic complexity, NPATH, and number of statement metric values calculated. The total number of library modules stored in the system was 485, and these made up the extra fanin values. The fanin for each library module only included calls from

the non-library modules, as information on calls between the library modules themselves aren't available.

The data for analysis was provided by the SMW metric data reports and by database queries. The reports were uploaded onto a Apple Macintosh microcomputer, and imported into spreadsheet and statistics packages. The tab-delimited nature of the reports made importing easy one the files had been transferred from the UNIX system, and will work with analysis packages on a variety of computer systems. The database queries developed a set of temporary tables and views within an SMW database that allowed comparisons of data for different metrics (see Section 7.7), and then the results of these queries were uploaded as well. The analysis packages were then used for the analysis and presentation of the report and query data.

7.2 Basic Statistical Analysis of Metric Values.

This section includes some basic information extracted from the SMW system about the metric values for all the programs. Table 7.2 gives the mean, mode, minimum, maximum, and total size of the sample for each metric over all the programs combined. The log of NPATH is included in order to transform the data into a better distribution than the pure NPATH values, as the NPATH values include a few very large numbers that skew the distribution.

Metric	Mean	Count	Min	Max	Mode
Fanin	2.577	2310	0	131	1
Fanout	3.118	1825	0	98	1
Fanout to Lib. Mods.	0.877	1825	0	28	0
McCabe (v(G))	6.487	1825	0	435	1
No. Statements	16.322	1825	0	1316	1
NPATH	2429195.657	1821	0	1671070000	1
log(1+NPATH)	1.167	1821	0	9.223	0.301

Table 7.2 Basic statistical information for each metric over all programs.

The frequency distributions for the total program data can be found in part 5 of Appendix D.

7.3 Empirical Evaluation of the NPATH metric.

This experiment had the aim of testing the correlation figures between the NPATH metric and several other metrics published in the paper describing the NPATH metric[77]. This experiment would use the SMW system in it's capacity for providing data for the evaluation of already established product metrics using metric values calculated for the test program data

[77]Nejmeh, B A; NPATH: A Measure of Execution Path Complexity and its Applications

set. These metric values would then be used in evaluating whether the trends claimed by the author of the NPATH metric were apparent in the test set of data supplied by SMW.

In the paper describing the NPATH metric the author compares metric values calculated from a set of 821 functions in a UNIX software application. The metrics chosen for comparison were the NPATH, McCabe's cyclomatic complexity (v(G)), a token count (TOKENS), and a metric based on the number of non-commentary source code lines (NCSL). Each of these metrics was evaluated for the individual functions in the author's test set. The NCSL metric was a count of the number of lines of code in a function that weren't white space or comments. A line can therefore consist of both declarative and non-declarative statements as well as punctuation. The token count metric is a count of all the lexical tokens in a function, including operator symbols, identifiers, keywords and punctuation such as semi-colons. This metric is commonly used as the basis for software science calculations. The published results of the correlation between the metrics can be seen in the Table 7.3.

	NCSL	TOKENS	v(G)	NPATH
NCSL	1.00	0.99	0.97	0.57
TOKENS	0.99	1.00	0.97	0.53
v(G)	0.97	0.97	1.00	0.56
NPATH	0.57	0.53	0.56	1.00

Table 7.3 Coefficients of determination published in NPATH paper (R-squared values).

From the data in the table it is clear that there is a strong linear relationship between then size of a function, measured in NCSL, the number of tokens, and it's cyclomatic complexity. Thus the larger the size of the module the greater it's cyclomatic complexity or token count is going to be. Nejmeh states that this relationship points to the fact that these three metrics, NCSL, tokens and cyclomatic complexity, are measuring the same program characteristics, namely the "lexical" complexity of the piece of software.

The lack of any strong relationship between the NPATH metric and the other metrics indicates that the NPATH metric measures something different, in this case "semantic" complexity, based on the number of execution paths in the function. Nejmeh states that the difference between the NPATH metric and the others doesn't mean that one sort of metric is better than any other, but the they measure different things, and both sorts are useful.

In the experiment metric values were supplied by the SMW metric collection tools for the NPATH, cyclomatic complexity, and number of statements metrics. The latter metric was used to determine the size of a module in much the same way as the NCSL metric was in Nejmeh's experiment.

The correlation coefficients were calculated for each set of metric values supplied for each

program in the test set, as well as for the total test set that included all the observations from all the different programs combined. These results were used to generate the coefficients of determination (R^2 values) that can bee seen in Tables 7.4 and 7.5.

Program	Number of Modules	No. Statements vs v(G)	No. Statements vs NPATH	v(G) vs NPATH
Bib	62	0.94	0.20	0.10
Bison	129	0.90	0.09	0.06
Chef	190	0.90	0.09	0.19
Compress	16	0.98	0.82	0.88
Flex	95	0.90	0.09	0.12
Prolog	115	0.92	0.15	0.13
Top	55	0.90	0.06	0.03
microEmacs	389	0.90	0.09	0.08
VN	122	0.93	0.08	0.08
WM	71	0.95	0.04	0.06
Xscheme	467	0.82	0.11	0.16
Yap	109	0.92	0.22	0.19

Table 7.4 Coefficients of determination for each program in the test set. (R-squared values)

Program	Number of Modules	No. Statements vs v(G)	No. Statements vs NPATH	v(G) vs NPATH
All Programs	1820	0.885	0.015	0.016

Table 7.5 Coefficients of determination for all the modules in the test set combined.

From these results the strong linear relationship that Nejmeh noted between the size of a module and it's cyclomatic complexity is clearly shown. It is more prominent at the individual program level but it still apparent when all the information from the individual programs is combined into a single data set.

Again the NPATH metric has a very low linear relationship with the module size and cyclomatic complexity. This value is much lower than the value shown by Nejmeh, but this could be due to increased NPATH values due to the macro expansion that occurs in the C preprocessor before the metric is calculated. Whether Nejmeh actually preprocessed the test application is unknown. Investigation into the effect that the removal of the macro expansion process would have upon the NPATH values is currently being researched further[78], but preliminary findings indicate that the correlation between cyclomatic complexity and NPATH does improve markedly if the macro expansions are left out.

The low NPATH correlation does reinforce Nejmeh's argument that NPATH measures different program characteristics than the other two sorts of metric, and that program size

[78]Garner, S, Churcher, N and Smith, P; An Empirical Investigation of NPATH; *To be published.*

measures and cyclomatic complexity are related in some way. Figure 7.1 shows the plot of the number of statements versus the cyclomatic complexity with the linear trend visible.

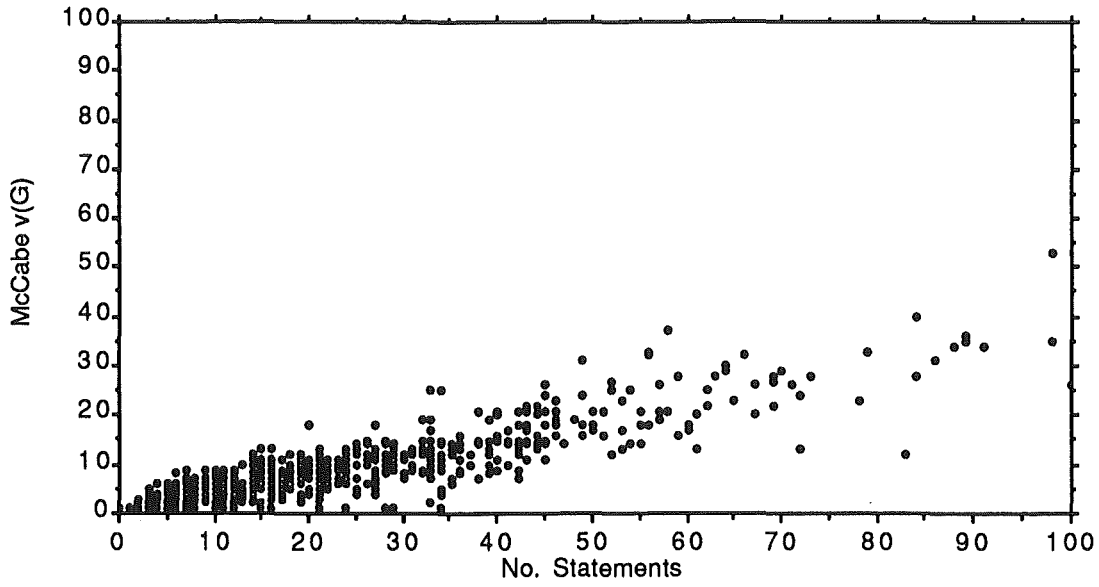


Figure 7.1 Graph of No. Statements vs McCabe's Cyclomatic Complexity.

7.4 Prediction of late life cycle complexity using early life cycle indicators.

The purpose of this experiment is to see if there is any correlation between the metrics that might be determined at an early stage in the design phase, as opposed to those determined at later stage. If this is the case then metric values can be calculated during the design phase of development that would allow the prediction of properties of the software product implemented.

A similar experiment was carried out by Troy and Zweben[79]. In their experiment they measured coupling, cohesion, complexity, size and modularity characteristics of 72 designs in an attempt to identify what features of the modules resulted in more source code modifications at the implementation of each design. They concluded that the primary area of influence at the implementation stage was based on the level of coupling between modules. Complexity and size also played roles in decreasing the quality of the implemented designs, but the cohesion factors were minimal in their results. Troy and Zweben concluded that 50-60% of the variance in the quality is due able to be explained by measurements of coupling, complexity and size.

The coupling measures used were based upon the use of structured design techniques which allowed for the measurement of data and control complexity in the interfaces between modules. The number of modules that a module influenced through it's scope of control

[79][Troy, D A and Zweben, S H; Measuring the Quality of Structured Designs

was also a factor, and this can be measured in part by fanout. In the cohesion measurements the fanin values of modules features, and in the complexity measurements fanout featured.

The structural fanin and fanout values calculated for the test set of programs to represent metrics that might be available in the design phase from product descriptions such as structure charts. These values were compared with the code metric values NPATH, number of statements and McCabe's cyclomatic complexity that had been calculated for the test set of programs.

From this it should be possible to look for certain trends that would allow us to make the desired predictions. The trends include the following:

• Is a module that is called by many different modules of a larger size or higher complexity than modules that are called by fewer modules? If a module is called by many modules it may be that it performs a single function that can be used by many different modules, much the same way as a library module is used (e.g. calculating the square root of a number). Or it may be that the module suffers from a lack of partitioning of functionality and based upon control information performs a variety of tasks. We would expect the first sort of module to be smaller and less complex than the latter module. Another possible explanation for the module's appearance is that it is a small module that is basically a switching statement, that for example might occur in the event loop of a user interface, and calls other modules based upon a the next event received.

• Does a module that calls a large number of other modules have a corresponding increase in size over a module with a lower fanout value? If the module is making a large number of different function calls it may be that it is performing a variety of tasks, with each task using a sub-set of the modules called. Or it may be that the module has a more complex execution path or control flow structure that is attributable to handling a variety of decisions that determine which if the modules are actually called during the function's current invocation.

Program	Number of Modules	Fanin vs No. Statements	Fanin vs v(G)	Fanin vs NPATH	Fanout vs No. Statements	Fanout vs v(G)	Fanout vs NPATH
Bib	62	0.006	0.006	0.002	0.548	0.388	0.238
Bison	129	0.000	0.001	0.000	0.139	0.069	0.003
Chef	190	0.016	0.016	0.004	0.461	0.355	0.022
Compress	16	0.194	0.236	0.236	0.870	0.914	0.863
Flex	95	0.003	0.006	0.005	0.259	0.324	0.542
Prolog	115	0.001	0.004	0.004	0.462	0.403	0.036
Top	55	0.001	0.002	0.002	0.340	0.203	0.026
microEmacs	389	0.002	0.002	0.000	0.207	0.178	0.017
VN	122	0.000	0.000	0.000	0.590	0.552	0.048
WM	71	0.009	0.011	0.015	0.289	0.307	0.142
Xscheme	467	0.000	0.000	0.001	0.442	0.323	0.012
Yap	109	0.000	0.003	0.002	0.229	0.179	0.078

Table 7.6 Coefficients of determintation by program between the early and late lifecycle metrics.

Program	Number of Modules	Fanin vs No. Statements	Fanin vs v(G)	Fanin vs NPATH	Fanout vs No. Statements	Fanout vs v(G)	Fanout vs NPATH
All Programs	1820	0.000	0.000	0.000	0.327	0.303	0.017

Table 7.7 Coefficients of determination for all programs between early and late lifecycle metrics.

The results in Tables 7.6 and 7.7 show a relationship between the fanout value of a module, the size of the module, and it's cyclomatic complexity. As the previous experiment demonstrated the strong correlation between size and cyclomatic complexity, this fact that both of them are related similarly to fanout is of little surprise. The NPATH values appeared to correlate erratically, and no overall correlation between fanout and NPATH can be seen. Fanout correlations with the size and cyclomatic complexity measures can account for up to 30% of the variability of the module's size and cyclomatic complexity. Assuming that the two latter measure can be used as estimates of the quality of the software product then the results appear to support Troy and Zweben's statement that coupling, complexity and size of designs, which all include fanout measurements, can be used to account for the variability of quality in the final product implemented.

Fanin appears to be of little value in estimating the metric values at the code level. This concurs with Troy and Zweben who found little correlation between the cohesion factors that fanin was part of, and the final quality of the implemented product.

7.5 Is there correlation between the Fanin-Fanout values?

In this study the relationship between the structural fanin and fanout values of a module were examined. Specifically, the study was to see if modules with higher fanin values had

correspondingly higher fanout values. This could be indicative of a lack of cohesion in modules resulting in an increase in the module's coupling. Modules with a high fanin might be performing several different functions within a module, which could cause more modules to be called by the module.

These fanin and fanout values for the individual programs, as well as the combined values for all the programs were examined, and the following information was obtained. Table 7.8 shows the R^2 coefficients of determination for fanin versus fanout for each program in the test set. Table 7.9 shows the R^2 value for all the programs combined. Both sets of data indicate that the variation in fanin values doesn't appear to account significantly for the values of the fanout values if a linear model of correlation is used. The distribution of all the fanin versus fanout values can be seen in the scatter plot Figure 7.2 and appears to be a form of hyperbolic function.

Most of the values have fanin and fanout values of 7 or less (95% of all fanin and 91% of all fanout). The modules with high fanout values and low fanin are typically high level control modules that are called only once or twice and delegate functionality to lower level subordinate modules. Modules with high fanin values and low fanout values typically occur at the lowest level of the program's structure and have high fanin due to being called by models at higher levels. These modules are the user-defined "library" modules that contain functionality required to be made available to all other modules, e.g. an error handler to be called in abnormal situations. The bulk of the modules appear with fanin and fanout combinations of 10 or less for each. These modules are the ones that exist in the central levels of the program's structure and are the sub-ordinates called by the high level control modules, and that might call several of the user-defined "library" modules as well. Modules that exist outside these groups, i.e. those with fanin- & fanout values both over 10, should be treated to examination as they fall neither into the control, "library" or normal sub-ordinate classes and the fanin-fanout values might be indicative of a lack of cohesion, or excessive coupling.

Program	No. Modules	Fanin vs Fanout (R-squared)
Bib	62	0.004
Bison	129	0.000
Chef	190	0.015
Compress	16	0.205
Flex	95	0.013
Prolog	115	0.010
Top	55	0.077
MicroEmacs	389	0.003
VN	122	0.002
WM	71	0.009
Xscheme	467	0.003
Yap	109	0.002

Table 7.8 Coefficients of determination for fanin vs fanout by program.

Program	No. Modules	Fanin vs Fanout (R-squared)
All Programs	1825	0.002

Table 7.9 Coefficients of determination for all programs combined.

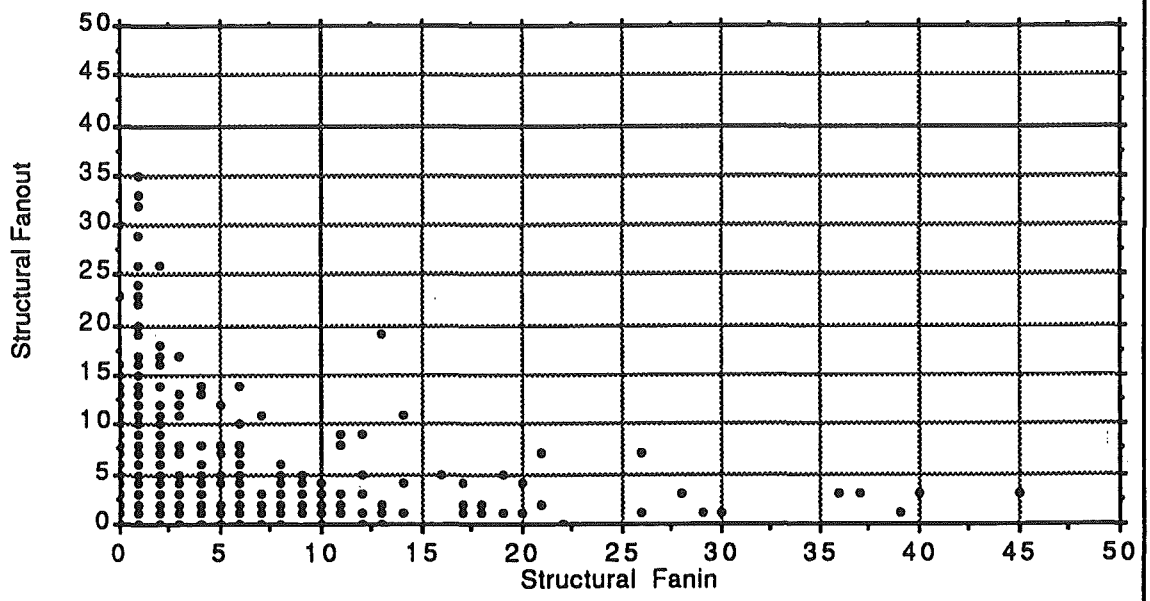


Figure 7.2 Scatter plot of fanin vs fanout for all programs.

7.6 The importance of the library call.

One of the aims of the SMW system is to allow the examination of relationships between the modules in a program. One such inter-module relationship is the contribution to the size of a

program's structure made by the library modules used by the program. Information about the call graph of a program stored in the SMW system, together with information in the type of module, such as whether it is a library module, allow this analysis to be performed. The objective of this case study was to obtain information on the following two subjects.

• The contribution to the overall size of a program, measured in modules, made by the library modules in each program. From this it can be seen whether the same proportion of each program is made up of library calls or whether the proportion of library calls is linked to the size of the program. In the last case it may be that there is a trend towards a maximum limit for the number of library modules a program uses.

• The contribution to the fanout values of each module made by library and non-library modules. From this an indication of the contribution of library calls to the complexity of the interface between modules in a program may be able to be obtained.

Firstly, the proportion of the program size made up by library modules was examined. Each of the programs in the test set was examined and the counts of library and non-library modules were obtained. These along with proportion of each program that each module type makes up are shown in Table 7.10 and are presented in Figure 7.3.

From the table and the graph it can be seen that the proportion of a program made up by library modules tends to decreases as the size of the program increases (R^2 value of 0.507). Intuitively one would expect a program to use a core set of library modules, which would then be combined in various different ways, in the user-defined modules. Once this core set of modules is defined, then the only growth in the development of a program would tend to be from the development of other user-defined modules The modules that use the core set within the program become used in the same way that library modules are used.

From Table 7.10 it may be noted that most programs in the test set have library module counts of a similar size regardless of the size or task of the program. This may indicate that in spite of difference between the applications each program does, that in C programs there is a common set of functions that are always needed, such as input/output function, particular string functions, and some mathematical functions. When extra library function are required by the specific program, such as the terminal handling library "curses" used by the window manager program WM, then the proportion of library modules goes up, as it does when the function of the software is closely bound with the computer hardware and operating system resources, as in the case of Compress and Top. Also it may be that a library of functions provides such a versatile range of functions that the developer doesn't have to write that many user defined functions to achieve the task.

	Total Number of Modules			Proportion by Module Type	
Program	Non-Library	Library	Both	Non-Library	Library
Bib	62	31	93	67%	33%
Bison	129	21	150	86%	14%
Chef	190	37	227	84%	16%
Compress	16	22	38	42%	58%
Flex	96	28	124	77%	23%
Prolog	117	44	161	73%	27%
Top	56	52	108	52%	48%
MicroEmacs	390	57	447	87%	13%
VN	122	54	176	69%	31%
WM	71	69	140	51%	49%
Xscheme	467	35	502	93%	7%
Yap	109	35	144	76%	24%

Table 7.10 Breakdown of modules by

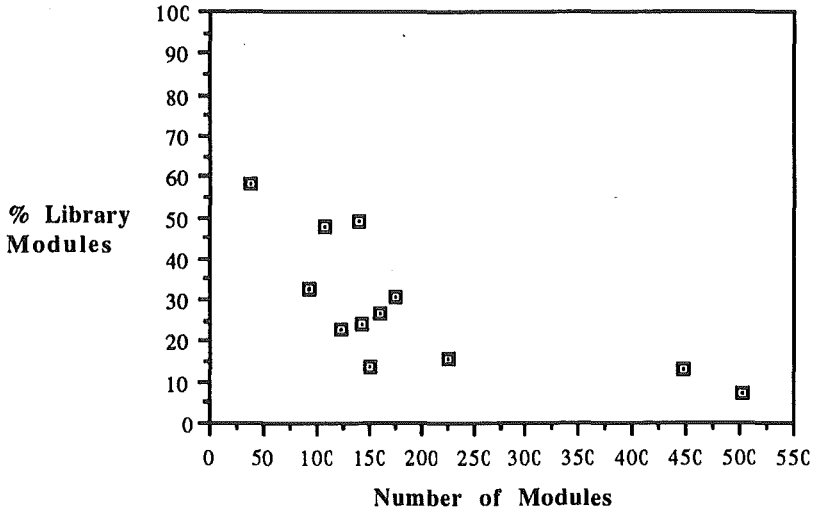


Figure 7.3 Graph of the number of modules in a program vs the proportion of those modules that are library modules.

The next part of this study involved determining the importance of the library module in the fanout values of modules. By isolating modules with high library module fanout components it may be possible to reduce the fanout. This could be achieved by combining various calls to library modules that are always called in the same sequence into another user-defined module. Calling this user-defined module would then add only one to the fanout value of the calling module, rather than adding the number of library modules.

The SMW system supplied fanout values, both the total fanout and the fanout to library modules, for the test set of programs. This information was used to determine how much of

the total fanout value in each program was due to library modules being accessed (Table 7.11). This information can be used to identify how important the library module fanout component is in the programs. Table 7.12 gives some supplementary information about the mean, mode, minimum and maximum fanout values for each of the programs, with Figure 7.4 giving a graphical comparison of the mean fanout values by program.

From these results it can be seen that the fanout to library modules contributes significantly to the overall fanout values of the modules. The fanout to library modules for the for all the modules in the test set combined accounted for about a quarter of the overall fanout values (Figure 7.5). Thus by isolating those modules with high fanout values to library calls for further investigation, and possibly partitioning into several modules with lower fanouts may improve the overall fanout values for modules reducing complexity and increasing the cohesion of modules.

			Library Calls		Non Library Calls	
Program	No. Modules	Total Fanout	Total Fanout	% Fanout	Total Fanout	% Fanout
Bib	62	207	113	54.59%	94	45.41%
Bison	129	348	117	33.62%	231	66.38%
Chef	190	603	118	19.57%	485	80.43%
Compress	16	58	39	67.24%	19	32.76%
Flex	96	251	73	29.08%	178	70.92%
Prolog	117	327	81	24.77%	246	75.23%
Top	56	188	124	65.96%	64	34.04%
MicroEmacs	390	1218	338	27.75%	880	72.25%
VN	122	574	278	48.43%	296	51.57%
WM	71	324	171	52.78%	153	47.22%
Xscheme	467	1253	67	5.35%	1186	94.65%
Yap	109	339	81	23.89%	258	76.11%

Figure 7.11 Components contributing to fanout by program.

		Library Calls				Non-Library Calls				Total Calls			
Program	No. Modules	Mean	Mode	Min	Max	Mean	Mode	Min	Max	Mean	Mode	Min	Max
Bib	62	1.82	0	0	14	1.52	0	0	9	3.34	2	0	18
Bison	129	0.91	N/A	0	5	1.71	0	0	14	2.7	1	0	16
Chef	190	0.62	0	0	7	2.55	1	0	29	3.17	1	0	29
Compress	16	2.44	1	0	14	1.19	0	0	9	3.63	1	0	23
Flex	96	0.76	0	0	4	1.85	1	0	21	2.62	1	0	24
Prolog	117	0.69	0	0	11	2.1	0	0	61	2.73	1	0	69
Top	56	2.21	1	0	28	1.14	0	0	30	3.36	2	0	58
MicroEmacs	390	0.87	0	0	8	2.26	0	0	95	3.12	1	0	98
VN	122	2.28	0	0	11	2.43	0	0	30	4.71	2	0	35
WM	71	2.41	0	0	12	2.16	0	0	23	4.56	0	0	30
Xscheme	467	0.14	0	0	9	2.54	1	0	23	2.68	1	0	24
Yap	109	1.52	0	0	9	2.37	1	0	33	3.11	1	0	33

Table 7.12 Basic fanout statistics for fanout components.

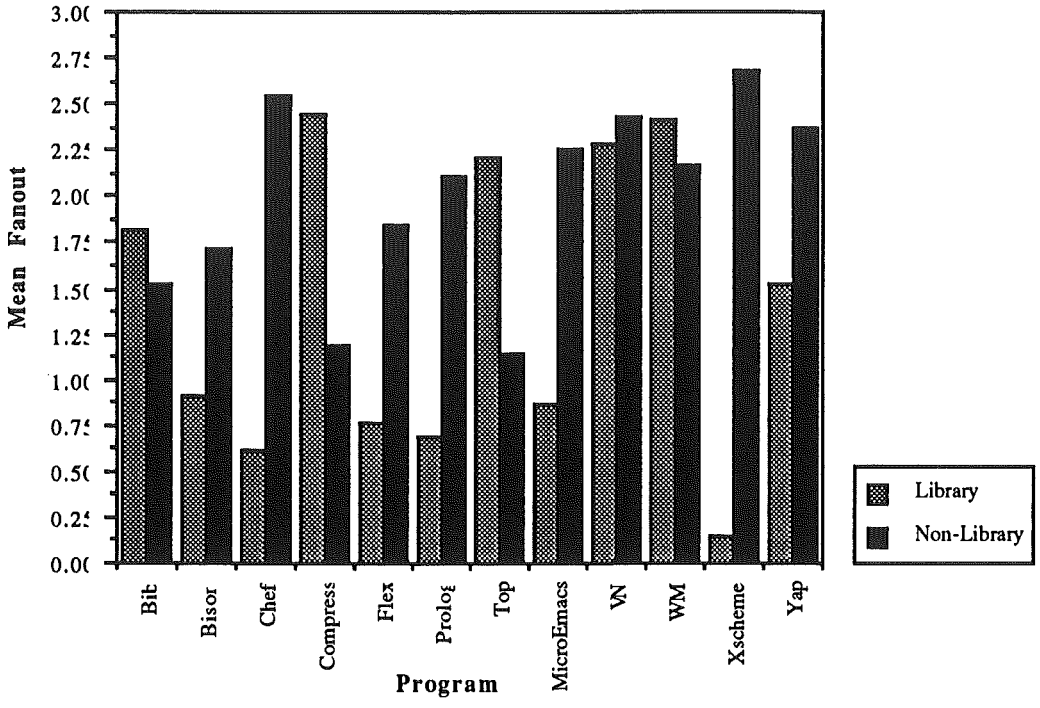


Figure 7.4 Mean fanout values library and non-library modules by program.

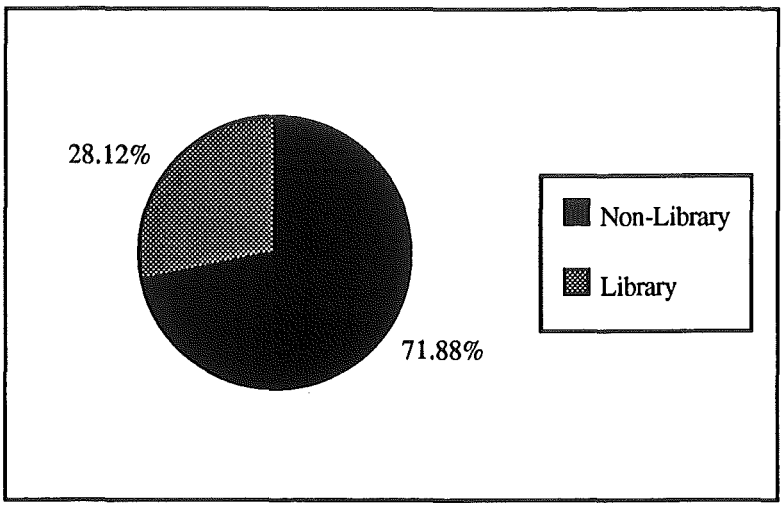


Figure 7.5 Proportions of fanout total contributed to by library and non-library modules for all programs.

	Non-Library Modules			Library Modules		
Program	Min. Fanin	Max. Fanin	Mean Fanin	Min. Fanin	Max. Fanin	Mean Fanin
Bib	0	11	1.516	1	15	3.645
Bison	0	45	1.791	1	39	5.571
Chef	0	36	2.553	1	23	3.189
Compress	0	3	1.188	1	7	1.773
Flex	0	12	1.854	0	13	2.607
Prolog	0	9	2.103	1	7	1.841
Top	0	3	1.143	1	14	2.385
MicroEmacs	0	40	2.256	1	85	5.93
VN	0	26	2.426	1	32	5.148
WM	0	14	2.155	1	11	2.478
Xscheme	0	131	3.099	1	11	1.943
Yap	0	20	2.367	1	7	2.314

Table 7.13 Fanin breakdown for library and non-library modules.

Table 7.13 shows the minimum, maximum and mean fanin values for the library and non-library modules in each of the programs in the test set. As is to be expected, the library modules tend to have a higher mean fanin than the non-library modules. However in some cases, such as in the programs Prolog and Xscheme, the mean fanin value for library modules is lower than the non-library module mean fanin. This can be indicative of a set of user-defined "library" modules within a program, that are called by many module Identification of these routines, once the interface for these "library" modules has been defined then the modules may be removed to a separate library for inclusion at link time. Another point raised by the high non-library fanin values is that inappropriate use may be being made of existing library modules available, and the developer has produced separate user-defined versions of these modules.

By using the fanin and fanout figures for non-library and library modules, developers could

build a picture of the use of library modules within the program, with indications of inappropriate use of library modules, of which modules might be better as separately compiled libraries, and the relative importance of the library modules used within the software.

7.7 Correlation of maximal cutoff points.

The proponents of various software metrics have suggested various "cutoff" or threshold values which if a function or module's metric value exceeds implies that it should be examined in order to reduce it's complexity. For the metrics cyclomatic complexity, NPATH, number of statements these values are 10[80], 200[81] and 50 respectively. For fanin and fanout the accepted maximum value is between 5 and 9[82], and values that were greater than 7 were used to represent these sets of values.

In this experiment modules from the test data with values greater than the accepted cutoff values for each metric were isolated, and then each set of modules was examined to see what modules had consistently high values for different metrics.

Each program data set was examined individually, and in combination with all the others. The results for the combination of the program data is described here, and the results of the individual programs may be found in Appendix D. The total size of the data set used for this experiment was 1825 modules. This didn't include the fanin values to library modules, but did include the large negative NPATH values due to the fact that if the values were so larger that they caused the value to become negative then they should be included in those NPATH values greater than 200 in value.

The results are presented in Tables 7.14 and 7.15. The first table (7.14) shows the total number of modules that exceeded the cutoff values for each metric. The second table (7.15) shows the frequency of those modules that have several metric values that exceed the cutoffs.

Metric	No. Statements > 50	McCabe > 10	NPATH > 200	Fanin > 7	Fanout > 7
No. Modules	99	270	222	87	154

Table 7.14 Total number of modules exceeding each metric threshold

[80]McCabe, T J; A Complexity Measure

[81]Nejmeh, B A; NPATH: A Measure of Execution Path Complexity and its Applications

[82]Page-Jones, M; The Practical Guide to Structured Systems Design; *Yourdon Press, New York*; 1980

No. Statements > 50	v(G) > 10	NPATH > 200	Fanin > 7	Fanout > 7	No. Modules	No. Metrics
•					0	1
			•	•	1	2
	•	•	•	•	1	4
•				•	1	1
•	•	•	•	•	1	5
		•	•		2	2
	•		•		3	2
•	•	•	•		3	4
	•	•	•		4	3
		•		•	8	2
•	•				9	2
•	•			•	11	3
	•			•	13	2
	•	•		•	20	3
•	•	•			30	3
		•			37	1
•	•	•		•	44	4
				•	54	1
	•				59	1
			•		72	1
	•	•			72	2

Table 7.15 Number of modules that exceed metric value thresholds by combinations of metrics.
(• indicates value exists at greater than threshold value).

From Table 7.14 it can be seen that the metric value that is most often exceeded is McCabe's cyclomatic complexity, followed in decreasing by NPATH, fanout, the number of statements, and then fanin.

From the information in the tables we can make the following observations about the particular metrics and their metric value cutoffs or thresholds.

The number of statements metric threshold appears to be possibly too large. Every instance of the threshold being exceeded is matched with at least one other metric threshold value being exceeded. Also the number of statements metric was matched by an exceeding cyclomatic complexity values in all case except two. These can be explained by the high correlation seen between the two metric in earlier experiments, and reinforces the idea that they might be measuring the same program characteristics.

Over eighty percent of the fanin values that exceed the fanin metric threshold of seven have no corresponding metric values in other metrics that exceed their respective thresholds. Again this supports the idea that fanin isn't a good indicator of complexity or size later in the development life cycle. However excessive fanin can be used to detect a high level of coincidental cohesion in modules.

Figure 7.6 shows the distribution of modules exceeding threshold values. Just over half of all the metrics that exceed a metric value only exceed one value. About a quarter exceed two metric values, and so on. Within each of those groupings there are sub-groupings that can be see in Table 7.15. For example of the single metric group 99 belong to the number of statements metric.

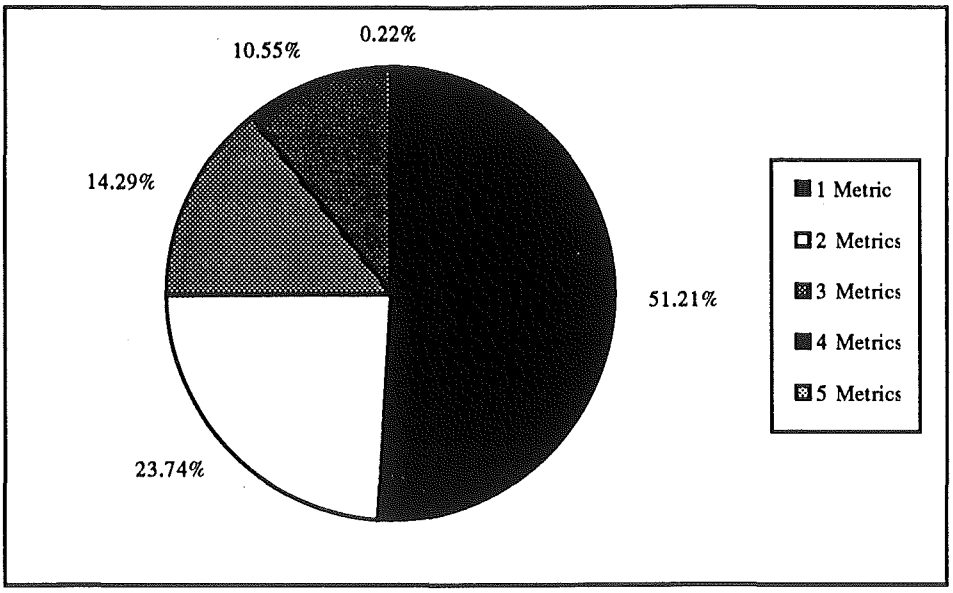


Figure 7.6 Proportions of modules exceeding metric value thresholds.

The correlation matrix for the metric values greater than the threshold values is shown in Table 7.16. The strong relationship between cyclomatic complexity and the number of statements is again shown, and the level of correlation between NPATH and cyclomatic complexity and number of statements has increased. Fanin shows a low correlation with any of the metrics.

	v(G)	NPATH	No. Statements	Fanin	Fanout
v(G)	1				
NPATH	0.043	1			
No. Statements	0.942	0.037	1		
Fanin	0.017	0.001	0.016	1	
Fanout	0.216	0.019	0.25	0.031	1

Table 7.16 Correlation matrix for metric values greater than threshold values. (R-squared values)

7.8 Conclusion of results.

The previous sections have shown a variety of analyses that SMW can provide information

for through both the SMW tool set and using query language scripts and interactive commands. The ability to import data supplied by the SMW reports into a variety of external tools is relatively easy, and allows the use of high quality presentation and analysis tools without the user having to write the tools themselves.

The studies themselves, along with the appendices containing metric data, have shown that the information supplied by the SMW system can be used in many ways. These include providing simple statistical information and answering questions about the importance of elements of program structure, through to evaluating the work does previously in the software metrics field.

Chapter 8 Summary and Conclusion.

This chapter contains a summary of the SMW system, including discussions on its uses, its strengths and weaknesses, and in what areas it can be enhanced. The SMW system is discussed within the context of software development and metric issues raised in the first three chapters.

In Chapter 1 the desire to produce quality software in order to reduce the cost of managing and maintaining software systems was discussed. Treating the software development cycle in similar way to engineering disciplines has run into problems due to the lack of quantitative measures and models to interpret those measures. Software metrics is endeavouring to establish models and measures that can produce the quantitative data required. Metrics development though has been hampered by the lack of historical data of a consistent nature that allows metrics programs and models to be developed and validated.

8.1 The SMW System's Current Status.

The SMW system was designed to allow a historical database of program and metric information to be collected, stored and made available for examination and manipulation. This would allow for verification and testing of existing metrics and their models, aid in the development of new models and metrics, and be able to be used as a method of storing an account of a program's development. The SMW sysem can be extended to by creating new tools that use the consistent access methods found in the data interface, and by adding new levels of abtraction in the database through the use of views and new tables. The SMW system has been and is currently being used for the collection of program and metric data and new tools, such as variants of the NPATH metric analysis tool, are being added.

Because the most promising application appears to be in applying metrics early in the software development life cycle, the SMW system is oriented towards storing the type of information that would be available during the system and detailed design phases of development. This is not to say that it is not applicable at an earlier or later phases, but rather that current support is not as great in terms of program information support in those phases. For example, the inter-module connections and data flows can be extracted from the source code of a program, but the internal structure of a module is not able to be stored. Thus SMW strengths in storing program data lie in it's ability to store relationships between modules such as caller-callee relationships and information flow in the form of parameters, global data objects, and module return values.

The ability to store multiple versions of multiple programs in an SMW database promotes the idea of setting up different databases for sets of similar programs, or the same program being developed several different ways. The program data is stored in the database so that the

software product it was extracted from is accessed only once. This means that the development process is disturbed less than if the product was accessed every time information about a relationship within the product was required. Instead the program data inside the SMW database may be accessed. This storage of of multiple versions also allows the maintenance on a program to be tracked.

Metric data is able to be stored equally well for all phases of the development process, both for module level metric values, and for program level metric values if a specially designated module identifier (normally '0') is used to represent these programs. Metric data of a non-numerical type is not stored, though data not of this type tends to be subjective in nature, such as classifications (e.g. "good", "bad", "very bad") or ranking (e.g. "best", "worst") which can be mapped onto numerical identifiers if desired.

Program and metric data within the SMW system is able to be accessed easily, through the query language monitors, the SMW user interface screens or through user written programs. All of these access the system through the data interface supported by the database management system, which enforces consistency of access methods and maintains data integrity within the SMW database. The availability of the query language from both the monitors and embedded inside high level language programs, means that data may be retrieved and manipulated in a ad-hoc fashion, as well as the query language being used in analysis tools.

Data inside an SMW system is able to be secured from access through the security measures present in the database management system, and through the use of unique integer identifiers which allows views of the data to be set up. This allows access to potentially sensitive program data to be stored and used by users who normally would not be allowed to access it. This broadens the amount of data that can be used for research purposes.

The SMW system allows for transferral of data between individual databases, and between SMW itself and other applications. In the latter case, a selection of textual reports produced by SMW can be imported into a range of statistical and numerical analysis packages for further processing, or directly from the database into a package if the user creates a tool to do so. The SMW Browser program does the latter when it sends scatter plot data to the GNUPlot plotting package.

The basic SMW tool set provides a range of tools for the collection of program and metric data, administration tools and a user interface for examining an SMW system database.

Program data collection allows the capture of the inter-module call relationships and module return values based upon the output from the standard UNIX tool "cflow" which analyses C, Lex, Yacc and assembler source files.

Metric data can be generated for McCabe's cyclomatic complexity, NPATH and number of statements metrics by analysis of the C source code product, as well as fanin and fanout information being able to be calculated from the program information stored in the SMW database. This last set of metrics (fanin-fanout) is at present the only language independent metrics available. These tools are examples of what sort of tools can be developed and new tools can be created by the user as the need arises.

Tools exist for the administration of the SMW database, including tasks such as adding and deleting metric information, and producing reports describing it's contents. The SMW Browser user interface allows users to examine the contents of the database, to perform structured queries, to access sub-systems of the database management system, such as the query language monitors and QBF ("Query-by-forms") query screens, and to generate plots and reports. The interface is design to have new functionality added to it in the way of new screens and menus but only if the user can program. If the user desires query forms can be set up in QBF which can be accessed from inside the browser by calling the QBF sub-system.

8.2 Drawbacks and Limitations of the SMW System.

The current SMW system has several limitations which will need to be addressed in the future.

Firstly, in spite of the database being designed for program information from the design phase the SMW system lacks tools that can examine a design notation and capture program information from it. At present all program information is captured from program source code which allows design information to be extracted but only after the design phase has finished. On a positive side this allows the examination of that design information to see what characteristics present in the design produced the implementation, but it does not give the ability to use the SMW system for prediction. The tools could be used in a maintenance environment though. The lack of a pseudocode or other design notation tool is not a great obstacle as a new tool for this can be added at a later date, in much the same way as a Pascal source code analysis tool could be added.

The quality of the program data at present is limited by the output from the utility "cflow". This utility provides information on the calls between functions, as well as basic information on function locations and return types. It doesn't however give us any information on data objects present in the functions, such as parameters and variables. However it does provide useful information on program structure, and another tool based on the C parser/grammar combination could be added at a later date to remedy this deficiency.

The C source analysis tools should be able to handle C source code with unexpanded macro

definitions. This would give a better picture of the source code as the programmer sees it, though if one tool does it then all the tools should so that the limitations of each tool are the same. This problem is not hard to overcome though and is currently being attended to.

The SMW database schema doesn't currently include the facility for the storage of the internal structure of modules. Thus for a metric that has to be calculated from the structure or contents of the module the software product has to be accessed, possibly disturbing the development process. However, if the recomputation of a metric is required then the product is probably going to be compiled or checked into a revision control system anyway, and if metric collection tools are embedded in those types of application (as mentioned below) then this does not pose a problem. Also the inter-module structure of a program may be represented in a generic way but the peculiarities of different programming languages intra-module structre would make this hard to implement in a generic way.

8.3 Possible Enhancements to SMW.

Most of the possible enhancements to the SMW system now lie in the area of increasing the set of tools and the ways in which they are used.

Firstly the number of metric data collection tools should be increased to build up a larger set of measures of program characteristics that can be compared. Candidates for inclusion in the new sets of metrics include Henry and Kafura's information flow metric, McCabe and Butler's structural design metric and Card and Agresti's architectural complexity metric. These are all design metrics and could be compared against the information available from the code metric tools.

Tools for the capture of program data at an earlier stage of development should also be concentrated on. These could be tied in with the existing computer-aided software engineering tools that allow the development of design documents, and especially those such as Oracle*CASE[83] that use a relational database for the support of their tools, which could be integrated to work with the SMW schema.

The program data and metric collection tools might also be incorporated into other tools that are used in development environments. For example, the metrics and program data collection tools for source code analysis might be incorporated into a compiler so that whenever the program or source files are compiled metric or program data can be captured. Another possibility is to integrate the collection tools into a revision control system, so that whenever a program file is checked into the system the metric or program data can be collected for the functions in that file.

The database scheme might be modified to be able to store information on the control

structure within the module. This would be useful as it would allow the access to the software product to be minimized for metrics collection. Difficulties with doing this would involve the storage of conditional expressions controlling the flow of control within the module, such as information on the order and nesting of the conditions.

The SMW administration tools for adding metric to and deleting metrics from the database could be integrated into the SMW user interface, though these options should only be available to authorized SMW system administrators.

8.4 Future Directions for the SMW System.

The SMW system has been established and has started to be used to evaluate metrics and to examine relationships between them (Chapter 7). The next stage for the SMW system is in the collection of a larger body of program and metric data from software products that are readily available to a wide range of researchers. From this data set a set of metric base lines can be developed for the comparison of metrics and the development of new models. The software analyzed should preferably be updated reasonably frequently so that new versions can be analyzed and the effects of modifications and enhancements to the software can be examined.

Preliminary studies have been started based upon the public domain version of Ingres, University Ingres version 8, in order to examine characteristics in a medium sized software system made up of a variety of programs, developed by a range of developers.

Ideally, the SMW system should be used to capture program and metric data from a large "real-world" application, from the design phase through to implementation, to allow the study of changes in the product, and to see if errors can be predicted before they happen. In order for this to happen the system must be modified to cope with design document analysis and more metric tools implemented to provide a greater range of measurements to be compared and modelled.

8.5 Conclusion.

The Software Metrician's Workbench system is a flexible and extensible environment for the collection and analysis of software product metric data. The use of a database management system as the underlying structure for the storage of metric and program data has provided a reliable form of storage. As well the database management system provides for the flexible and consistent manipulation of the data in the SMW system through the use of query language tools.

New tools can easily be added to the SMW system, and existing tools modified without the physical structure of the storage facility required. Tools have been created for program data

capture, metric data capture and analysis, administration and user interfacing.

SMW offers the ability to export data and reports for further analysis should this be necessary, as well as the provision for free-form and forms based queries.

A body on metrics data is being developed for the examination of different metrics and metric models. This body or corpus of metric data is designed to be used as a set of benchmark values that other further research will be used with. Thus the SMW system will supply what has been so sorely needed in the field of software metrics - a body of historical and test data that can be used for the validation and evaluation of existing and new metrics and models. From this better models and metrics can be developed and deficient ones updated or discarded.

The SMW system has worked well in collecting and providing metric data for analysis. The analysis performed so far has been used to test the claims for some metrics, as well as provide insight into the structure of programs, particularly with library modules, and the implementation of tools, both at a query language and high level language levels, has proved to be relatively easy once them measurement is defined clearly.

Positive feedback was received from a recently presented paper on the SMW system[84] (Appendix F) and the SMW system has become an ongoing research project, is currently being used to research into the nature of the NPATH metric[85].

It is hoped that the examination and analysis of the metric data will lead to the identification of properties in software at all phases in the software development life cycle, but particularly at early phases, that will improve the detection of errors in the product and overly complex parts of the product. Thus the quality of the software will increase, as will it's reliability and it's cost to develop and maintain will be reduced.

---

[83]Oracle*CASE, *Oracle Corporation, 20 Davis Drive, Belmont, California 94002, U.S.A.*

[84]Garner, S R and Churcher, N A; A Software Metrician's Workbench; *Proc. 12th N.Z. Computer Conference;* 41-48; 1991

[85]Garner, S, Churcher, N and Smith, P; An Empirical Investigation of NPATH; *To be published.*

Bibliography.

Albrecht, A J and Gaffney, J E Jnr; Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation; *IEEE Trans. Software Engineering;* Vol SE-9; No. 6; 639-648; 1983

Baker, A L, Bieman, J M, Fenton, N, Gustafson, D A, Melton, A C and Whitty, R; A Philosophy for Software Measurement; *J. Systems Software;* Vol. 12; 277-281; 1990

Basili, V R and Hutchens, D H; An empirical Study of a Syntactic Complexity Family; *IEEE Trans. Software Engineering;* Vol SE-9; No. 6; 664-672; 1983

Basili, V R and Perricone, B T; Software Errors and Complexity: An Empirical Investigation; *Communications of the ACM;* Vol. 27; No. 1; 42-52; 1984

Basili, V R and Turner, A J; Iterative Enhancement: A practical Technique for Software Development; *IEEE Trans. Software Engineering;* Vol. SE-1; No. 4; 390-396; 1975

Bastani, F B and Iyengar, S S; The Effect of Data Structures on the Logical Complexity of Programs; *Communications of the ACM;* Vol. 30; No. 3; 250-259; 1987

Behrens, C A; Measuring the Productivity of Computer Systems Development Activities with Function Points; *IEEE Trans. Software Engineering;* Vol SE-9; No. 6; 648-652; 1983

Belady, L A and Evangelesti, C J; System Partitioning and It's Measure; *Systems Software*; Vol. 2; 23-39; 1981

Benyon-Tinker, G; Complexity measures in an evolving large system; Proc. ACM Workshop Quant. Software Models; 117-127; 1979

Bieman, J A, Baker, A L, Clites, P N, Gustafson, D A and Melton, A C; A Standard Representation of Imperative Language Programs for Data Collection and Software Measures Specification; *J. Systems and Software;* Vol. 8; 13-37; 1988

Botting, D; Comments on Tripp's Notations; *ACM Sigsoft;* Vol. 14; No.1; 83; 1989

Bowen, J B; Module Size: A Standard or Heuristic; *J. Systems Software;* Vol. 4; 327-332; 1984

Brandl, D L; Quality Measures in Design, Finding Problems before coding; *ACM Sigsoft;* Vol. 15; No. 1; 68-72; 1990

Bush, M E and Fenton, N E; Software Measurement: A Conceptual Framework; *J. Systems Software;* Vol. 12; 223-231; 1990

Campbell, R H and Terwilliger, R B; The SAGA Approach to Automated Project Management; *Lecture Notes in Computer Science;* No. 244; 142-155;

Card, D N and Agresti, W W; Resolving the Software Science Anomaly; *J. Systems and Software;* Vol. 7; 29-35; 1987

Card, D N and Agresti, W W; Measuring Software Design Complexity; *J. Systems and Software;* Vol. 8; 185-197; 1988

Cavano, J P; Software Reliabilty Measurement: Prediction, Estimation, and Assessment; *J. Systems and Software;* Vol. 4; 269-275; 1984

Chamberlin, D D and Boyce, R F; "SEQUEL: A Structured English Query Language"; Proc 1974 ACM SIGMOD Workshop on Data Description, Access and Control; 1974

Conte, S D, Dunsmore, I and Shen H E; Software Engineering Metrics and Models; *Benjamin Cummings, Menlo Park;* 1986

Cote, V, Bourque, P, Oligny, S and Rivard, N; Software Metrics: An Overview of Recent Results; *J. Systems and Software;* Vol. 8; 121-131; 1988

Coulter, N S, Cooper, R B and Solomon, M K; Information-theoretic Complexity of Program Specifications; *The Computer Journal;* Vol. 30; No. 3; 1987

Coulter, N S; Software Science and Cognitive Psychology; *IEEE Trans. Software Engineering;* Vol SE-9; No. 2; 166-171; 1983

Coupal, D and Robillard, P N; Factor Analysis of Source Code Metrics; *J. Systems Software;* Vol. 12; 263-269; 1990

Crawford, S G, McIntosh, A A and Pregibon, D; An Analysis of Static Metrics and Faults in C Software; *J. Systems Software;* Vol. 5; 37-48; 1985

Date, C J; An Introduction to Database Systems; *Addison-Wesley Publishing Company*; Vol. 1; 4th Ed.; 209-232; 1986

Date, C J; A Guide to Ingres; *Addison-Wesley Publishing Company*; 1987

Davis, J S and LeBlanc, R J; A Study of Applicability of Complexity Measures; *IEEE Trans. Software Engineering;* Vol. 14; No. 9; 1366-1372; 1988

Dehnad, K; Software Metrics from a Users Perspective; *J. Systems Software;* Vol 13.; 111-115; 1990

DeMarco, T; Controlling Software Projects: Management, Measurement and Estimation; *Yourdon Press, NY, U.S.A*; 1983

Dittrich, K R, Gotthard, W and Lockemann, P C; DAMOKLES - A Database System for Software Engineering Environments; *Lecture Notes in Computer Science;* No. 244; 353-371; 1986

Evangelist, W M; Software Complexity Metric Sensitivity to Program Structuring Rules; *J. Systems Software;* Vol. 3; 231-243; 1983

Fenton, N E; Software Metrics: Theory, Tools and Validation; *Software Engineering Journal;* 65-78; 1990

Fenton, N E and Kaposi, A A; Metrics and Software Structure; *Information and Software Technology;* Vol. 29; No. 6; 301-320; 1987

Fenton, N and Melton, A; Deriving Structurally Based Software Measures; *J. Systems Software;* Vol. 12; 177-187; 1990

Frewin, G D; Metrics in Procurement-a Discussion Paper; *Proc. Centre for Software Reliability Conference: Measurement for Software Control and Assurance;* 89-102; 1987

Garner, S and Churcher, N A; A Software Metrician's Workbench; *Proc. 12th N.Z. Computer Conference;* 41-48; 1991

Ghezzi, C and Jazayeri, M; Programming Language Concepts; *John Wiley & Sons Inc, New York;* 1982

Glass, R L; Software Metrics: of Lightning Rods and Built-Up tension; *J. Systems and Software;* Vol. 10; 157-158; 1989

Halstead, M H; Elements of Software Science; *Elsevier North-Holland, New York*; 1977

Harrison, W; MAE: A Syntactic Metric Analysis Environment; *J. Systems and Software;* Vol. 8; 57-62; 1988

Harrison, W; A Foreword to the Special issue on Using Software Metrics; *J. Systems Software;* Vol. 13; 87-88; 1990

Harrison, W and Cook, C; A Method of Sharing Industrial Software Complexity Data; *SIGPlan Notices;* Vol. 20; No. 2; 42-51; 1985

Heitkoetter, U, Helling, B, Nolte, H and Kelly, M; Design Metrics and Aids to their Automatic Collection; *Information and Software Technology;* Vol. 32; No. 1; 79-87; 1990

Henry, S and Kafura, D; The Evaluation of Software Systems' Structure using Quantitative Software Metrics; *Software- Practice and Experience;* Vol. 14; No. 6; 561-573; 1979

Henry, S and Kafura, D; Software Structure Metrics Based on Information Flow; *IEEE Trans. Software Engineering;* Vol. SE-7; No. 5; 510-518; 1981

Henry, S, Kafura, D and Harris, K; On the Relationship Among Three Software Metrics; *ACM SIGmetrics;* Spring; 81-88; 1981

Henry, S and Lewis, J; Integrating Metrics into a Large-scale Software Development Environment; *J. Systems Software;* Vol. 13; 89-95; 1990

Ince, D; Software Metrics; *Proc. Centre for Software Reliability Conference: Measurement for Software Control and Assurance;* 27-62; 1987

Ince, D; Software Metrics: Introduction; *Information and Software Technology;* Vol. 32; No. 4; 297-303; 1990

Ince, D C and Hekmatpour, S; An Approach to Automated Software Design Based on Product metrics; *Software Engineering Journal;* 53-56; 1988

Johnson, D; Graphical Program Notations: On Tripps Survey, The Past, and the Future;

ACM Sigsoft; Vol. 14; No. 5; 78-79; 1989

Kafura, D and Reddy, G; The Use of Software Complexity Metrics in Software Maintenance; *IEEE Trans. Software Engineering;* Vol. SE-13; No. 3; 335-343; 1987

Karimi, J and Konsynski, B R; An Automated Software Design Assistant; *IEEE Trans. Software Engineering;* Vol. 14; No. 2; 194-210; 1988

Kearney, J K, Sedlmeyer, R L, Thompson, W B, Gray, M A and Adler, M A; Software Complexity Measurement; *Communications of the ACM;* Vol. 29; No. 11; 1044-1050; 1986

Kernighan, B and Ritchie, D; The C Programming Language; *Prentice-Hall, N.J*; 1978

Kernighan, B and Ritchie, D; The C Programming Language; *Prentice-Hall, N.J.*; 2nd Ed.; 1988

Kitchenham, B; Software Reliabilty and Metrics; *Software Engineering Journal;* 2; 1990

Kitchenham, B A and Linkman, S J; Design Metrics in Practice; *Information and Software Technology;* Vol. 32; No. 4; 304-310; 1990

Kitchenham, B A and McDermid, J A; Software Metrics and Integrated Project Support Environments; *Software Engineering Journal;* 58-64; 1986

Kitchenham, B A, Pikard, L M and Linkman, S J; An Evaluation of some Design Metrics; *Software Engineering Journal;* 50-58; 1990

Knuth, D E; An Empirical Study of FORTRAN Programs; *Software- Practice and Experience;* Vol. 1; 105-135; 1971

Kokol, P, Ivanek, B and Zumer, V; Software Effort Metrics: How to Join Them; *ACM Sigsoft;* Vol. 13; No. 2; 55-57; 1988

Lassez, J L, van der Knijff, D, Shepherd, J and Lassez, C; A Critical Examination of Software Science; *J. Systems and Software;* Vol. 2; 105-112; 1981

Laughery, K R Jnr and Laughery, K R Snr; Human Factors in Software Engineering: A Review of the Literature; *J. Systems Software;* Vol. 5; 3-14; 1985

Leach, R J; Software Metrics and Software Maintenance; *Software Maintenance: Research and Practice;* Vol. 2; 133-142; 1990

Lecciso, R, Mainetti, S and Morasca, S; Software Metrics: A Critical Evaluation and an application to Pascal.; *Microprocessing and Microprogramming;* Vol. 18; 605-616; 1986

Lewis, J A and Henry, S M; On the Benefits and Difficulties of a Maintainability Via Metrics Methodology; *Software Maintenance: Research and Practice;* Vol. 2; 113-131; 1990

Li, E Y; A Measure of Program Nesting Complexity; *National Computer Conference; 1987;* Vol. 56; 531-538; 1987

Li, H F and Cheung, W K; An Empirical Study of Software Metrics; *IEEE Trans. Software Engineering;* Vol. SE-13; No. 6; 697-708; 1987

Lohse, J B and Zweben, S H; Experimental Evaluation of Software Design Principles: An Investigation into the Effect of Module Coupling on System Modifiabilty; *J. Systems Software;* Vol. 4; 301-308; 1984

MacDonell, S; An Examination of Techniques for the Measurement of Programmer Productivity and Software Complexity; *New Zealand Journal of Computing;* Vol. 2; No. 1; 33-38; 1990

MacDonell, S and Sallis, P; Establishing Complexity Correlations Between Pseudocode and Program Source Code; *New Zealand Journal of Computing;* Vol. 2; No. 1; 39-44; 1990

Marti, R W; Integrating Database and Program Descriptions Using an ER-Data Dictionary; *J. Systems and Software;* Vol. 4; 185-195; 1984

McCabe, T J; A Complexity Measure; *IEEE Trans. Software Engineering;* Vol. SE-2; No. 4; 308-320; 1976

McCabe, T J and Butler, C W; Design Complexity Measurement and Testing; *Communications of the ACM;* Vol. 32; No.12; 1415-1425; 1989

Mills, E E; Software Metrics: SEI Curriculum Module SEI-CM-12-1.1; *Carnegie Mellon University;* 1988

Myers, G J; An extension ot the cyclomatic measure of program complexity; ACM SIGPLAN Notices; Vol. 12; No. 10; 61-64; 1977

Myrvoid, A; Data Analysis for Software Metrics; *J. Systems Software;* Vol. 12; 271-275; 1990

Nakata, S and Yamazaki, G; ISMOS: An Experimental Database-Oriented Tool Generator; *J. Systems and Software;* Vol. 4; 219-238; 1984

Narayanaswamy, K and Scacchi, W; A Database Foundation to Support Software System Evolution; *J. Systems and Software;* Vol. 7; 37-49; 1987

Navlakha, J K; Software Productivity Metrics: Some Candidates and Their Evaluation; *National Computer Conference; 1986;* 1986

Navlakha, J K; A Survey of System Complexity Metrics; *The Computer Journal;* Vol. 30; No. 3; 233-238; 1987

Nejmeh, B A; NPATH: A Measure of Execution Path Complexity and its Applications; *Communications of the ACM;* Vol. 31; No. 2; 188-200; 1988

Norcio, A F and Chmura, L J; Designing Complex Software; *J. Systems Software;* Vol. 8; 165-184; 1988

Oman, P W and Cook, C R; Design and Code Traceability Using a PDL Metrics Tool; *J. Systems Software;* Vol. 12; 189-198; 1990

Page-Jones, M; The Practical Guide to Structured Systems Design; *Yourdon Press, New York*; 1980

Petrova, E and Veevers, A; Role of Non-Stochastice-Based Metrics in Quantification of Software Reliability; *Information and Software Technology;* Vol. 32; No. 1; 71-78; 1990

Pickard, L M; Analysis of Software Metrics; *Proc. Centre for Software Reliability Conference: Measurement for Software Control and Assurance;* 155-180; 1987

Pintelas, P E and Kallistros, V; An Overview of Some Software Design Languages; *J. Systems and Software;* Vol 10.; 125-138; 1989

Poore, J H; Derivation of Local Software Quality Metrics (Software Quality Circles); *Software- Practice and Experience;* Vol. 18; No.11; 1017-1027; 1988

Prather, R E; An Axiomatic Theory of Software Complexity Measure; *The Computer Journal;* Vol. 27; No. 4; 340-347; 1984

Prather, R E; On Hierachical Software Metrics; *Software Engineering Journal;* 42-45; 1987

Prather, R E; Comparison and Extension of Theories of Zipf and Halstead; *The Computer Journal;* Vol, 31; No. 3; 248-252; 1988

Prosser, D F; American National Standard X3.159-1989, Programming Language C; *American National Standards Institute,New York, N.Y.*; 1989

Ramamurthy, B and Melton A C; A Synthesis of Software Science Measures and the Cyclomatic Number; *IEEE Trans. Software Engineering;* Vol. 14; No. 8; 1116-1121; 1988

Redish, K A and Smyth, W F; Evaluating Measures of Program Quality; *The Computer Journal;* Vol. 30; No. 3; 228-232; 1987

Redmond, J A and Ah-Chuen, R; Software Metrics - a Users Perspective; *J. Systems Software;* Vol 13.; 97-110; 1990

Reed, K; Practical Software Engineering Environments: Report on the ACM SIGSOFT/SIGPLAN Software Engineering Symposium; *ACM Sigsoft;* Vol. 12; No. 1; 56-62; 1987

Reynolds, R G; Metrics to Measure the Complexity of Partial Programs; *J. Systems Software;* Vol. 4; 75-91; 1984

Reynolds, R G; Metric-based Reasoning about Pseudocode Design in the Partial Metrics System; *Information and Software Technology;* Vol. 29; No. 9; 497-502; 1987a

Reynolds, R G; The Partial Metrics System: Modeling the Stepwise Refinement Process using Partial Metrics; *Communications of the ACM;* Vol. 30; No. 11; 956-963; 1987b

Reynolds, R G; The Partial Metrics System: A Tool to Support the Metrics Driven Design of Pseudocode Programs; *J. Systems and Software;* Vol. 9; 287-295; 1989

Robillard, P N; Schematic Pseudocode for Program Constructs and its Computer Automation by Schemacode; *Communications of the ACM;* Vol. 29; No.11; 1072-1089; 1986

Robillard, P N; On the Evolution od Graphical Notations for Program Design; *ACM Sigsoft;* Vol. 14; No. 1; 84-88; 1989

Robillard, P N and Boloix, G; The Interconnectivity Metrics: A New Metric Showing how a Program is Organized; *J. Systems and Software;* Vol. 10; 29-39; 1989

Rodriguez, V and Tsai, W T; A Tool for Discriminant Analysis and Classification of Software Metrics; *Information and Software Technology;* Vol. 29; No.3; 137-150; 1987

Rook, P; Controlling Software Projects; *Software Engineering Journal;* 7-16; 1986

Roper, R M F and Smith, P; A Software Tool for Testing JSP Designed Programs; *Software Engineering Journal;* 46-49; 1987

Ross, N; The Collection and Use of Data for Monitoring Software Projects; *Proc. Centre for Software Reliability Conference: Measurement for Software Control and Assurance;* 125-154; 1987

Samson, W B, Nevill, D G and Dugard, P I; Predictive Software Metrics based on a Formal Specification; *Information and Software Technology;* Vol. 29; No. 5; 242-248; 1987

Shen, V Y, Conte, S d and Dunsmore, H E; Software Science Revisited: A Critical Analysis of the Theory and its Empirical Support; *IEEE Trans. Software Engineering;* Vol SE-9; No. 2; 155-165; 1983

Shen, V Y, Yu, T, Thebaut, S M and Paulsen, L R; Identifying Error-Prone Software - An Empirical Study; *IEEE Trans. Software Engineering;* Vol. Se-11; No. 4; 317-323; 1985

Shepperd, M; A Critique of Cyclomatic Complexity as a software metric; *Software Engineering Journal;* 30-36; 1988a

Shepperd, M; An Evaluation of Software Product Metrics; *Information and Software Technology;* Vol. 30; No. 3; 177-188; 1988b

Shepperd, M; Design Metrics: An Empirical Analysis; *Software Engineering Journal;* 3-10; 1990a

Shepperd, M; Early Life-Cycle Metrics and Software Quality Models; *Information and Software Technology;* Vol. 32; No. 4; 311-316; 1990b

Shepperd, M and Ince, D; Metrics, Outlier Analysis and the Software Design Process; *Information and Software Technology;* Vol. 31; No. 2; 91-98; 1989

Soupos, P, Goutas, D, Christodoulakis, C and Zaroliagis, C; The GRASPIN DB - A Software Development Environment Database; *ACM Sigsoft;* Vol. 12; No. 1; 63; 1987

Stonebraker, M R, Wong, E, Kreps, P and Held, G D; The Design and Implementation of INGRES; *ACM TODS*; Vol. 1; No. 3; 1976

Szulewski, P A, Whitworth, M H, Buchan, P and DeWolf, J B; The Measurement of Software Science Parameters in Software Designs; *ACM SIGmetrics;* Spring; 89-94; 1981

Tripp, L L; A Survey of Graphical Notations for Program Design - An Update; *ACM Sigsoft;* Vol. 13; No. 4; 39-44; 1988

Troy, D A and Zweben, S H; Measuring the Quality of Structured Designs; *J. Systems and Software;* Vol. 2; 113-120; 1981

Waguespack, L J and Badlani, S; Software Complexity Assessment: An Introduction and Annotated Bibliography; *ACM Sigsoft;* Vol. 12; No. 4; 52-71; 1987

Weyuker, E J; Evaluating Software Complexity Measures; *IEEE Trans. Software Engineering;* Vol. 14; No. 9; 1357-1365; 1988

Weiss, D M and Basili, V R; Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory; *EEE Trans. Software Engineering;* Vol. 11; 157-168; 1985

Wilson, C and Osterweil, L J; Omega - A Data Flow Analysis Tool for the C Programming Language; *IEEE Trans. Software Engineering;* Vol, SE-11; No. 9; 832-838; 1985

X3H2 (American National Standards Database Committee). American National Standards Database Language SQL: Working Draft. Document X3H2-85-1 (December 1984).

Yau, S S and Collofello, J S; Some Stability Measures for Software Maintenance; *IEEE Trans. Software Engineering;* Vol. SE-6; No. 6; 545-552; 1980

Yau, S S and Collofello, J S; Design Stability Measures for Software Maintenance; *IEEE Trans. Software Engineering;* Vol. SE-11; No. 9; 849-856; 1985

Yau, S S and Tsai J J P; A Survey of Software Design Techniques; *IEEE Trans. Software Engineering;* Vol. SE-12; No. 6; 713-721; 1986

Yin, B H and Winchester, J W; The establishment and use of measures to evaluate the quality of structured designs; Proc. ACM Software Qual. Ass. Workshop, 45-52, 1978

Yu, T J, Nejmeh, H E, Dunsmore, H E and Shen, V Y; SMDC: An Interactive Software Metrics Data Collection and Analysis System; *J. Systems and Software*; Vol. 8; 39-46; 1988

Appendices.

Overview of Appendices.

Appendix A SMW Database Definitions.

Appendix B SMW User Guide.

Appendix C NPATH Product Metric.

Appendix D SMW Test Program Data Set.

Appendix E C Lexical Scanner and Parser.

Appendix F SMW Paper.

Appendix G Sample Output Plot from GNUPlot

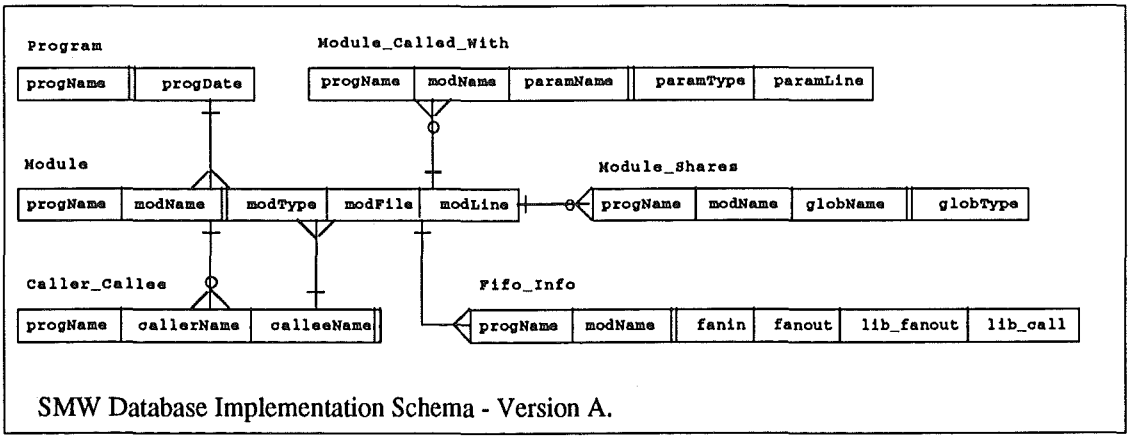
Appendix A. SMW Database Descriptions.

This appendix gives definitions for the tables present in the three different versions of the SMW database implemented. The type of data stored in each column is given, along with it's corresponding physical definition for the Ingres database management system. Entity-relationship diagrams are given for each of the database versions prior to the table definitions. The first section of the appendix gives the sizes of the text fields common various of the tables.

Sizes for Text Attributes in the SMW Database Tables.

Text Attribute	Bytes	Notes
fileName	50	
metName	80	
metProg	50	
modName	33	
progDate	25	(if being accessed through ESQL or EQUEL)
progName	50	
progLoc	255	(as defined by MAXPATHLEN in <sys/param.h> header file
typeName	40	
objName	33	

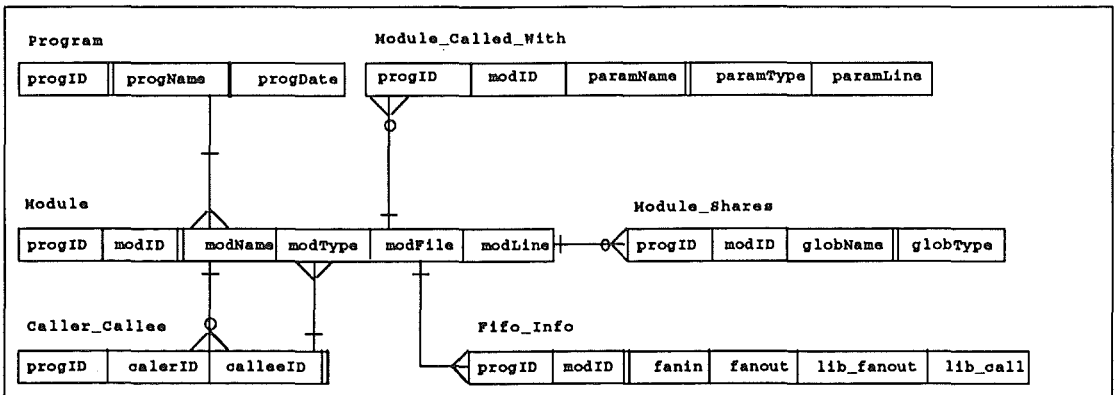
SMW Database - Version A.



SMW Database Implementation Schema - Version A.

Table	Attribute	Type of Data	Ingres Data Type	Contents
Caller_Callee	progName	text	text	Program name
	callerName	text	text	Name of calling module
	CalleeName	text	text	Name of called module
Fifo_Info	progName	text	text	Program name
	modName	text	text	Module name
	fanin	integer	i4	No. of module called by
	fanout	integer	i4	No. of modules called
	lib_fanout	integer	i4	No. of library module called
	lib_call	integer	i1	Library module flag
Module	progName	text	text	Program name
	modName	text	text	Module name
	modType	text	text	Data type of return value
	modFile	text	text	Source file module defined in
	modLine	integer	i4	Line in source file module defined at
Module_Called_With	progName	text	text	Program name
	modName	text	text	Module name
	paramName	text	text	Parameter name
	paramType	text	text	Data type of parameter
	paramLine	integer	i4	Line in source file defined at
Module_Shares	progName	text	text	Program name
	modName	text	text	Module name
	globName	text	text	Name of global data object
	glodType	text	text	Data type of global data object
Program	progName	text	text	Program name
	progDate	date	date	Date and time program info captured

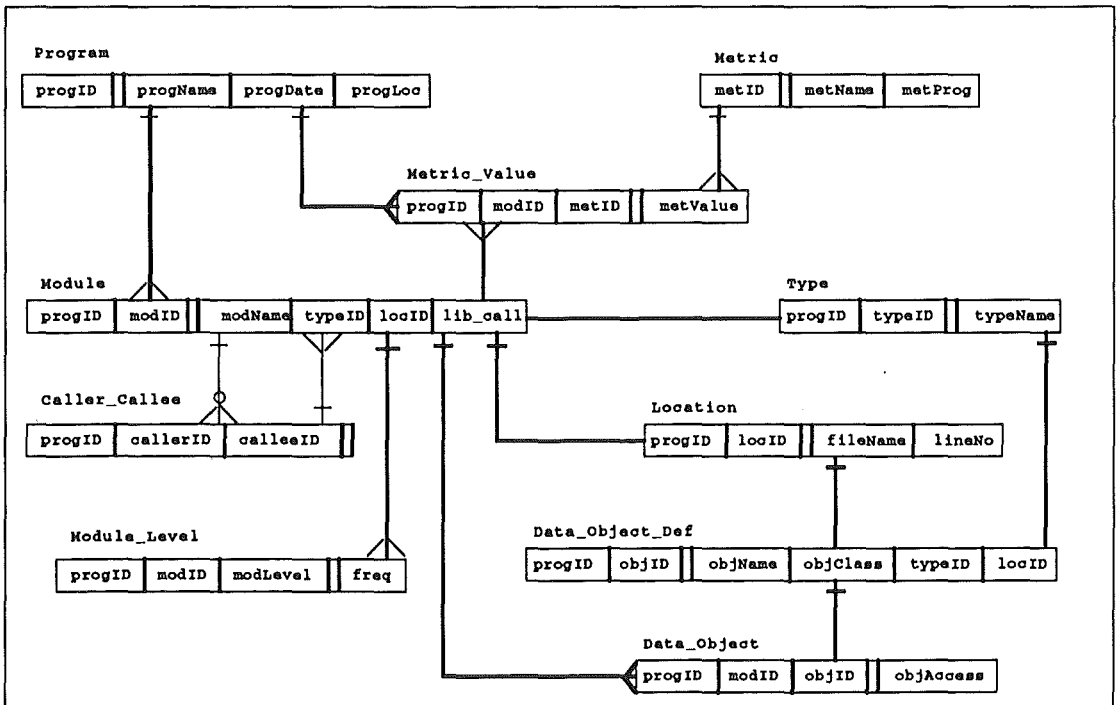
SMW Database - Version B.



SMW Database Implementation Schema - Version B.

Table	Attribute	Type of Data	Ingres Data Type	Contents
Caller_Callee	progID	integer	i4	Unique program identifier
	callerID	integer	i4	Calling module identifier
	calleeID	integer	i4	Caller module identifier
Fifo_Info	progID	integer	i4	Program identifier
	modID	integer	i4	Module identifier
	fanin	integer	i4	No. of module called by
	fanout	integer	i4	No. of modules called
	lib_fanout	integer	i4	No. of library module called
	lib_call	integer	i1	Library module flag
Module	progID	integer	i4	Program identifier
	modID	integer	i4	Module identifier
	modName	text	text	Module name
	modType	text	text	Data type of return value
	modFile	text	text	Source file module defined in
	modLine	integer	i4	Line in source file module defined at
Module_Called_With	progID	integer	i4	Program identifier
	modID	integer	i4	Module identifier
	paramName	text	text	Parameter name
	paramType	text	text	Data type of parameter
	paramLine	integer	i4	Line in source file defined at
Module_Shares	progID	integer	i4	Program identifier
	modID	integer	i4	Module identifier
	globName	text	text	Name of global data object
	glodType	text	text	Data type of global data object
Program	progID	integer	i4	Program identifier
	progName	text	text	Program name
	progDate	date	date	Date and time program info captured

SMW Database - Version C.



SMW Database Implementation Schema - Version C.

Table	Attribute	Type of Data	Ingres Data Type	Contents
Caller_Callee	progID	integer	i4	Unique program identifier
	callerID	integer	i4	Calling module identifier
	calleeID	integer	i4	Caller module identifier
Data_Object	progID	integer	i4	Program identifier
	modID	integer	i4	Module identifier
	objID	integer	i4	Data object identifier
	objAccess	integer	i1	Object access flag (0=unused,1=read from, 2=written to,3=read & written)
Data_Object_Def	progID	integer	i4	Program identifier
	objID	integer	i4	Data object identifier
	objName	text	text	Data object name
	objClass	integer	i1	Class of object (0=unknown,1=local variable, 2=global variable,3=file,4=parameter)
	typeID	integer	i4	Data type identifier
	locID	integer	i4	Identifier for object definition
Location	progID	integer	i4	Program identifier
	locID	integer	i4	Location identifer
	fileName	text	text	Name of source file
	lineNo	integer	i4	Line number
Metric	metID	integer	i4	Metric identifier
	metName	text	text	Metric name
	metProg	text	text	Name of metric collection tool
Metric_Value	progID	integer	i4	Program identifier
	modID	integer	i4	Module identifier
	metID	integer	i4	Metric identifier
	metValue	floating point	f8	Metric value
Module	progID	integer	i4	Program identifier
	modID	integer	i4	Module identifier
	modName	text	text	Module name
	typeID	integer	i4	Identifier for data type of return value
	locID	integer	i4	Identifier for module definition location
	lib_call	integer	i1	Library module flag
Program	progID	integer	i4	Program identifier
	progName	text	text	Program name
	progDate	date	date	Date and time program info captured
	progLoc	text	text	Location of source files
Type	progID	integer	i4	Program identifier
	typeID	integer	i4	Module identifier
	typeName	text	text	Data type name

Appendix B. SMW User Guide.

The following appendix contains a guide for users of the SMW system, and explains the use of the SMW Browser user interface program to collect program and metric data, to retrieve data, and to create reports. Information is also provided on how to use the SMW system administration tools and the formats of the reports available.

A

User Guide

for the

Software Metrician's

Workbench

System

Stephen R. Garner. (1991)

Table of Contents

A User Guide to the SMW System

1. Introduction.

The Software Metrician's Workbench (SMW) system has a prototype user interface built using the Ingres Forms and Menus tools. The aim of this document is to provide the user of the SMW system with a description of the functions offered by this interface and descriptions of how to access these functions. As well, a description of how to use the various tools and programs that make up the SMW system from the command line is given.

This guide is structured in the following way. The first section covers all the tasks that must be done before the user can start to use the SMW interface. The second section will cover the various tasks that the user may wish to do, such as creating a new database or generating a report. The third section covers each of the forms found in the SMW system and gives an account of what is begin displayed on each form and what options are available to the user from the form. The last section covers using the SMW system without the SMW interface, and how to extend the system, for example adding a new metric collection program. Finally there are appendices which discusses various technical and administrative aspects of the SMW system.

2. Getting Started.

This section covers what the user has to do before running the SMW system.

2.1 Setting up the environment.

Before the SMW system can be used several tasks need to be done.

Firstly the user must be set up as a valid Ingres user. This should be done by the system administrator of the user's computer system. Without this being done the user will not be able to create, destroy or access any Ingres databases, and therefore won't be able to use the SMW system. This process will also include setting up any environment variables that Ingres requires, such as *TERM_INGRES* which should be set up to be the same as the user's terminal type. For example,

```
setenv TERM_INGRES vt102k
```

The second thing to be done is to set up the user's UNIX environment variables that SMW uses. This involves creating one new environment variable, *SMWPATH*, and modifying the *PATH* environment variable. *SMWPATH* should be set to the directory where the various SMW programs exist. For example,

```
setenv SMWPATH /usr/local/smw
```

The *PATH* environment variable should have the location of the SMW programs added to it as well, so that the shell can find the programs when it needs to run them.

2.2 Starting Up the SMW System.

Having completed the tasks above the user may now start up the SMW system. To do this the user types the following at the shell prompt,

```
smw
```

and the user should be presented with the title screen shown in Figure 2.1.

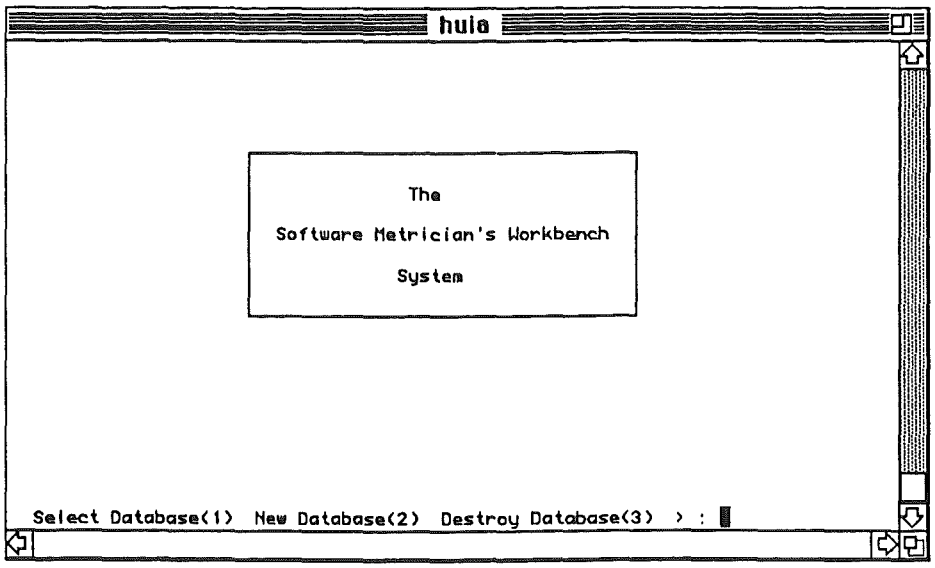


Figure 2.1 SMW Title Screen.

At the bottom of the screen there will be a list of menu options. These can be selected either by pressing the function key corresponding to the number beside the menu choice or by pressing the Ingres menu key and then typing in as much of the option name as is required to distinguish it from the other options and pressing return.

If, on any menu the menu option you select is called

`End`

then no further action taken on the menu and you will be returned to the previous menu.

Selecting the option

`Quit`

from the menu at the SMW title screen will cause you to exit from the SMW system and return to the shell that you ran the SMW system from.

A diagram of the SMW system menu structure is given in Appendix B.

3. Using the SMW System Through the Forms Interface.

3.1 About this section.

This section describes procedures for a selection of tasks that the users will normally want to perform most often. These tasks include:

- Creating a new SMW database.
- Destroying an existing SMW database.
- Putting data into the database such as program and metric data.
- Getting data out of the database in the form of reports, plots and queries.

Each of these tasks will be described as a series of steps that the user work through in order to complete the task. In order for the user to use the SMW system a database must be created and then the basic program information has to be collected before the metric data may be collected and used.

An overview of the SMW Forms Interface command structure can be found in Appendix B.

3.2 Creating a new database.

Before the user can use the SMW system an SMW database must be created. To do this perform the following steps.

1. Start up the SMW system by typing

   ```
   smw
   ```

   at the shell prompt. This should cause the SMW title screen to be displayed (Figure 2.1.).

2. Select the menu option

   ```
   New Database
   ```

3. Enter the name of the database and press the return key.

After this the SMW system will create a new database, and then the user will be returned to the title screen menu. If the return key was pressed without a database name

being entered then the user will be returned to the menu and no database will be created.

3.3 Destroying an existing database.

In order for the user to destroy a database that is finished with the following steps must be performed.

NOTE: Once the database has been destroyed all the data in it will be lost forever. Only use this command if you understand what it does and it's ramifications.

1. Start up the SMW system by typing

   `smw`

   at the shell prompt. This should cause the SMW title screen to be displayed (Figure 2.1.).

2. Select the menu option

   `Destroy Database`

3. Enter the name of the database and press the return key.

4. The user will then be asked whether to continue with the destruction process. If the user types in

   `y` or `Y`

   or any text starting with either of those characters (e.g. "yes" or "Yes") then the database and all it's contents will be destroyed. If the user types anything else then nothing will happen and the user will be returned to the title screen menu.

If the return key was pressed without a database name being entered then the user will be returned to the menu and no database will be destroyed.

3.4 Getting data into the database.

There is three sorts of data that need to be entered into the SMW database. Firstly there is the data that describes the basic program structure of a version of a program that you want to process. Secondly there is the metrics data that will be calculated and stored in the database. And thirdly there is the information on the metrics programs themselves. The collection of the last category of data will be discussed in Section 5.2.2.

3.4.1 Collecting the basic program data.

The basic program data is data that contains the name of the program, it's date entered into the system, the names of all it's modules, the program structure and other data such as the data type returned by the module when it's called.

To collect this data follow the steps below.

1. Start up the SMW system by typing

   `smw`

   at the shell prompt. This should cause the SMW title screen to be displayed (Figure 2.1.).

2. Select the menu option

   `Select Database`

3. Enter the name of the database and press the return key. After a while the SMW program information screen will be displayed (Figure 3.2.). If no name is entered then nothing will happen and the user will be returned to the title screen menu.

4. From the program information menu select the option

   `New Program`

   This will display a sub-menu with three options on it (Figure 3.3.). The `UNIX Shell` option is discussed later in Section 4.2.

5. Select the menu option

   `Metrics Shell`

   This should cause the user to be prompted for the pathname of the directory where the source files of the program are located. If this is known the user should enter the pathname and press return, otherwise the user should simply press return. The Metrics Shell screen should then be displayed (Figure 3.4), with a two column list of the files either in the directory specified in the last step by the user, or those in the current directory.

6. Using the Metrics Shell commands described in Section 4.7, the user should select the files for analysis, and the select the menu option

   `Analyser`

The analyser clear the screen and run. This can take a long time for a large program with many files. When the analyser is finished the user will be prompted to press any key, and upon doing so will be returned to the SMW program information screen with the new program sub-menu being displayed.

If the user selected the menu option

End

while in the Metrics Shell the user will be returned to the SMW program information screen and no program data will be input into the database.

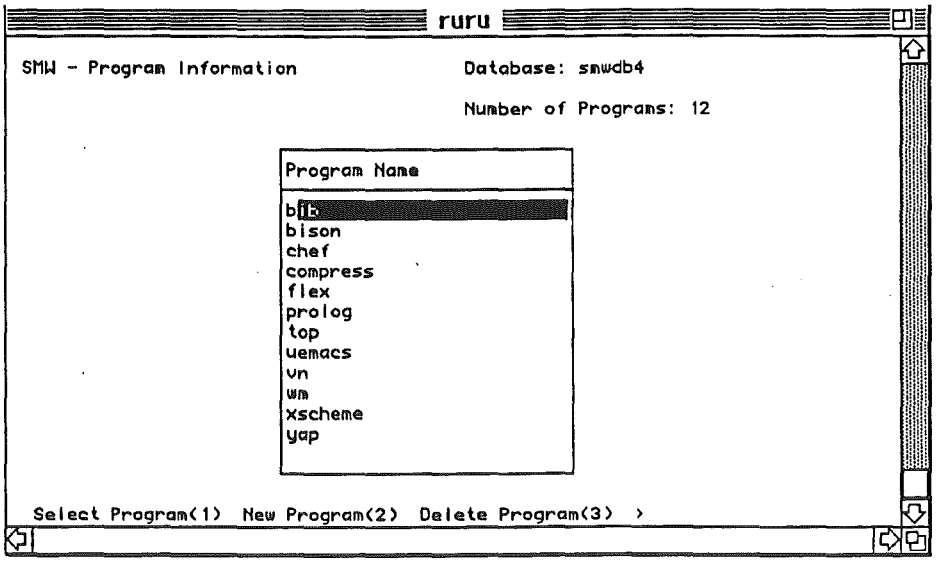


Figure 3.2 SMW Program Information Screen.

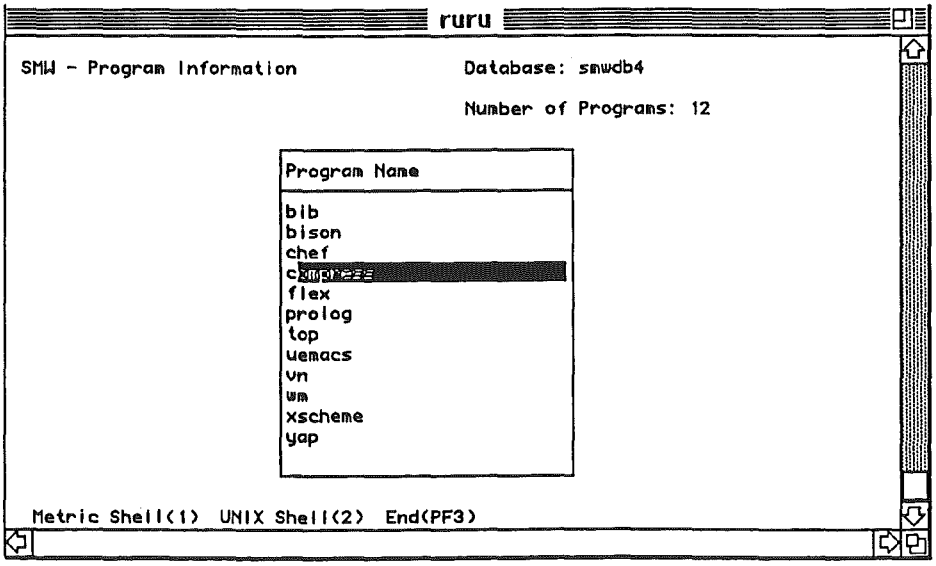


Figure 3.3 SMW Program Information Screen – New Program Sub-Menu

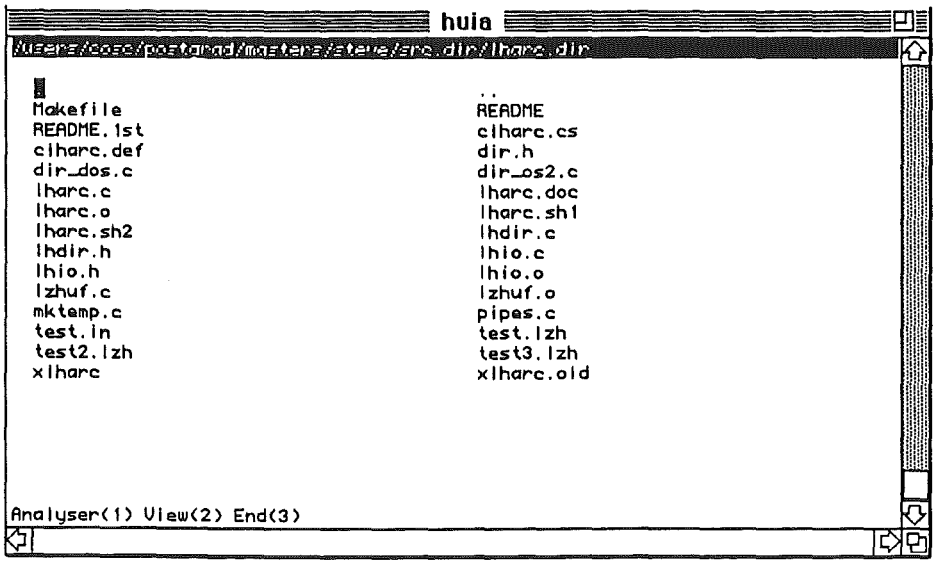


Figure 3.4 SMW Metrics Shell Screen.

3.4.2 Collecting the metric data.

Having collected the basic program data, metric values for the program or programs stored in the database may be calculated and stored in the database. The procedure for doing this is as follows.

1. The user should start up the SMW system and selected a database. This should cause the SMW Program Information screen to be displayed.

2. Move the cursor to the program to have the metric values calculated for and select the menu option

   `Select Program`

   This should bring up the SMW Program Version Information screen.

3. Select the menu option

   `Calculate Metric`

   This will bring up a box on the screen with a list of metrics in it.

4. Move the cursor to the metric you want to calculate and select the menu option

   `Select`

   If the metric has already been calculated the nothing will happen.

5. Select the menu option

   `End`

   to return to the SMW Program Version Information screen.

3.5 Getting data out of the SMW system

Data can be extracted from the SMW system in a variety of ways. It can be extracted in the form of reports, where the requested information is output to a report file, in the form of plots, and in the form of interactive queries.

3.5.1 Reports.

A variety of reports can be generated from the various screens in the SMW system, allowing information that the current screen deals with to be output to a text file for processing at a later time by the user. For example the user could save a report of all the metric values calculated for a particular metric for a program version and then format the report in a word processor or spreadsheet and then print it out. A variety of reports are now discussed and the procedures for generating them will be described.

Program information reports.

These reports are available from the SMW Program Information screen (Figure 3.3.). They contain information on what programs, and versions of programs, are currently stored in the database currently being processed. There are two types of report able to be created.

The first report, Only Program Names, is a sorted list of the names of the programs in the database. This report file will have the format described in Appendix A, Section A1.

The second report, Program Names and Versions, is a sorted list of all the names of the programs in the database, with a sorted list of all the versions of a program after that respective program's name. This report file will have the format described in Appendix A, Section A2.

To generate these reports the user must first have created a database (Section 3.2), and collected the basic program information (Section 3.4.1) for at least one program. Having done this the report desired can be created using the following procedure.

1. Select from the SMW Program Information screen menu the menu option

   `Report`

   This will display a sub-menu (Figure 3.5) with two reporting menu options on it.

2. Select the type of report you want to generate by choosing the appropriate menu option.

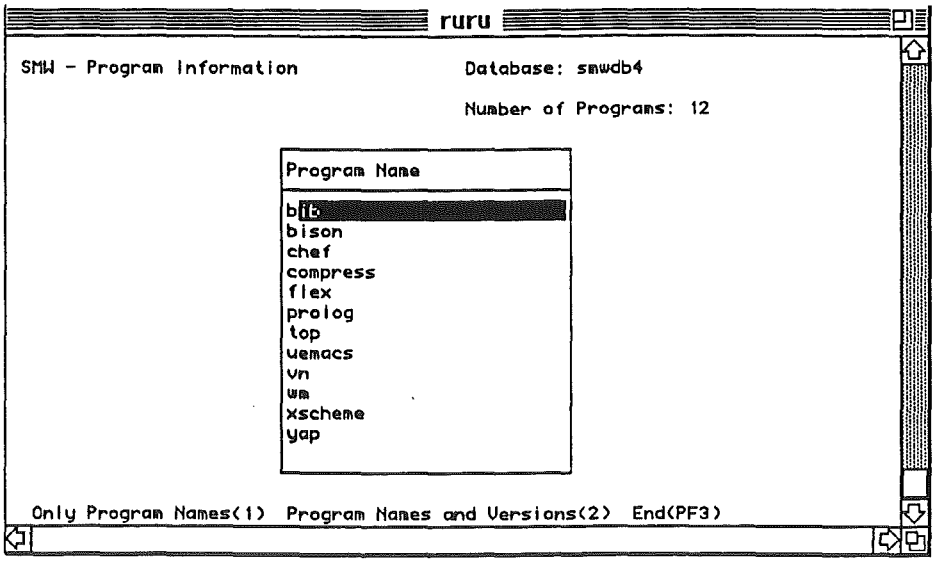


Figure 3.5 SMW Program Information Screen - Report Sub-menu.

3. When prompted the user should enter the name of the file to output the report to and press return. If no file name is entered then no report will generated and the user will be returned to the sub-menu where the report type was selected from.

4. When the report has been created the user will be returned to the sub-menu where the report type was selected from and can then select another (or the same) report option or can select

   `End`

   and return to the SMW Program Information screen menu.

Program version reports.

Reports are available for versions of programs. These reports provide information on metrics that have been calculated for versions of a program. A report can be generated for all the metrics calculated for a program version or just a single metric. For a report to be generated, the basic program information must have been collected, and at least one metric calculated for a program version.

The first report, All Metrics, is a table consisting of a sorted list of all the modules in the selected version with all the metric values generated for each module. This report file will have the format described in Appendix A, Section A3.

If the module has no value for a metric then there will be no value at the position of that metric value.

The second report, Selected Metric, is a sorted list of all the modules in the selected version with the specified metric value for each of those modules. This report file will have the format described in Appendix A, Section A4.

The following procedures can be used to generate these reports.

All Metrics

1. At the SMW Program Information screen move the cursor to the name of the program that the version you want to generate a report is of.

2. Select from the SMW Program Information Screen menu the menu option

   `Select Program`

   This will display the SMW Program Version Information screen (Figure 3.6.).

3. Move the cursor to the version of the program to have the report generated from.

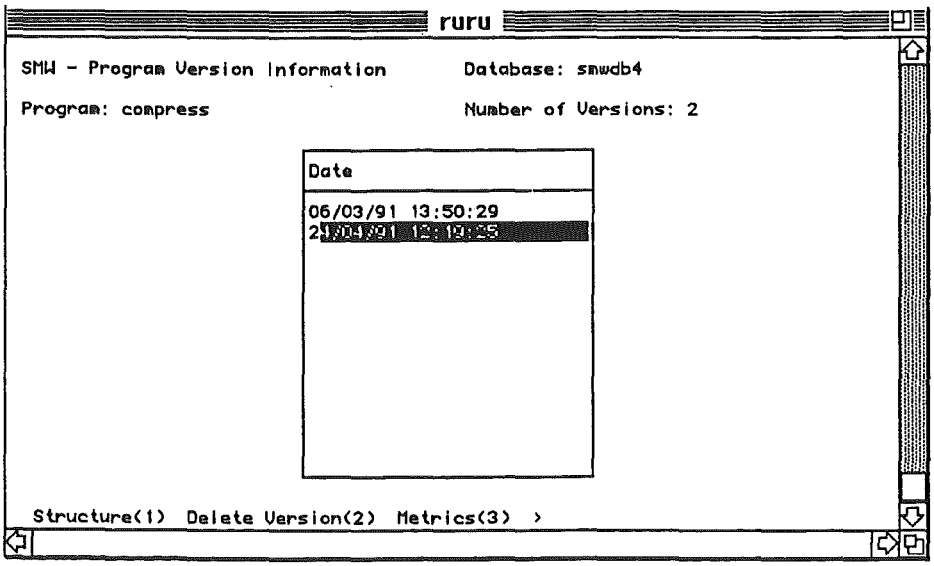


Figure 3.6 SMW Program Version Information Screen.

4. Select from the SMW Program Version Information screen menu the menu option

   `Report`

   This will display a sub-menu (Figure 3.7) with two reporting menu options on it.

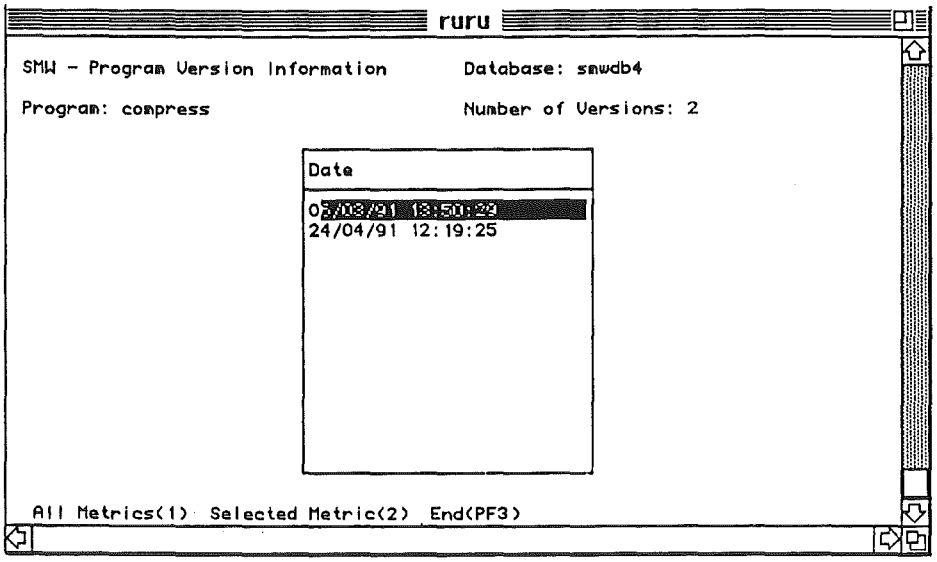


Figure 3.7 SMW Program Version Information Screen – Report Sub-menu.

5. Select the menu option

   `All Metrics`

6. When prompted the user should enter the name of the file to output the report to and press return. If no file name is entered then no report will generated and the user will be returned to the sub-menu where the report type was selected from.

7. When the report has been created the user will be returned to the sub-menu where the report type was selected from and can then select another (or the same) report option or can select

   `End`

   and return to the SMW Program Version Information screen menu.

Selected Metric

Follow steps 1 through 4 above, and then

5. Select the menu option

   `Selected Metric`

   If have been one or more metrics calculated then the user will be presented with a list

of metric names, and a new sub-menu (Figure 3.8.), otherwise nothing will happen.

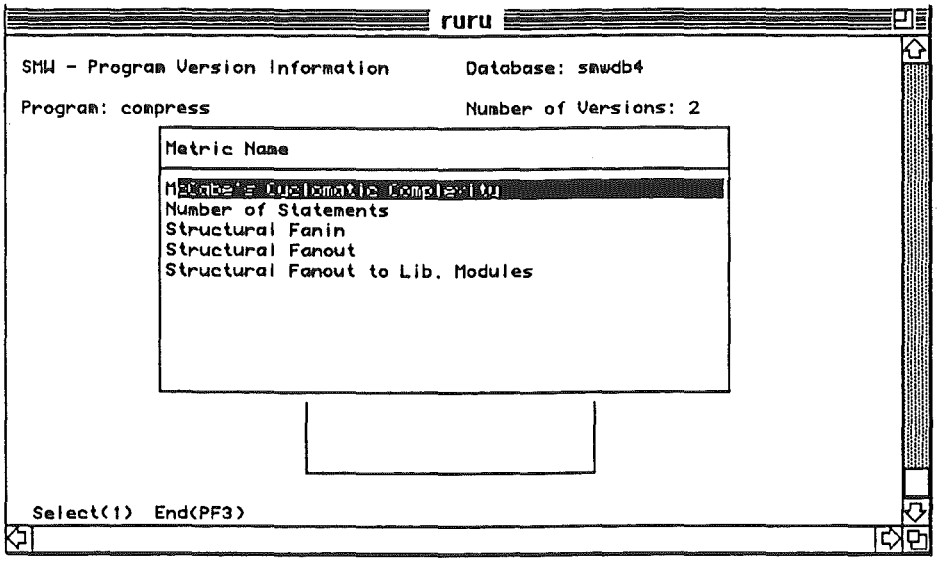


Figure 3.8 SMW Program Version Information Screen – Select Metric Report Sub-menu.

6. Move the cursor to the desired metric's name, and then choose the menu option

   `Select`

   If the user chooses the menu option

   `End`

   then nothing will happen and the user will be returned to the previous sub-menu.

7. When prompted the user should enter the name of the file to output the report to and press return. If no file name is entered then no report will generated and the user will be returned to the sub-menu where the report type was selected from.

8. When the report has been created the user will be returned to the sub-menu where the report type was selected from and can then select another (or the same) report option or can select

   `End`

   and return to the SMW Program Version Information screen menu.

Structure reports

Structure reports provide information on the inter-module structure of a program version. At a simple level they contain lists of what modules are in the program, and at a more complex level they describe the inter-connections between modules in the program version

selected. These reports are created from the SMW Module Information screen (Figure 3.9.).

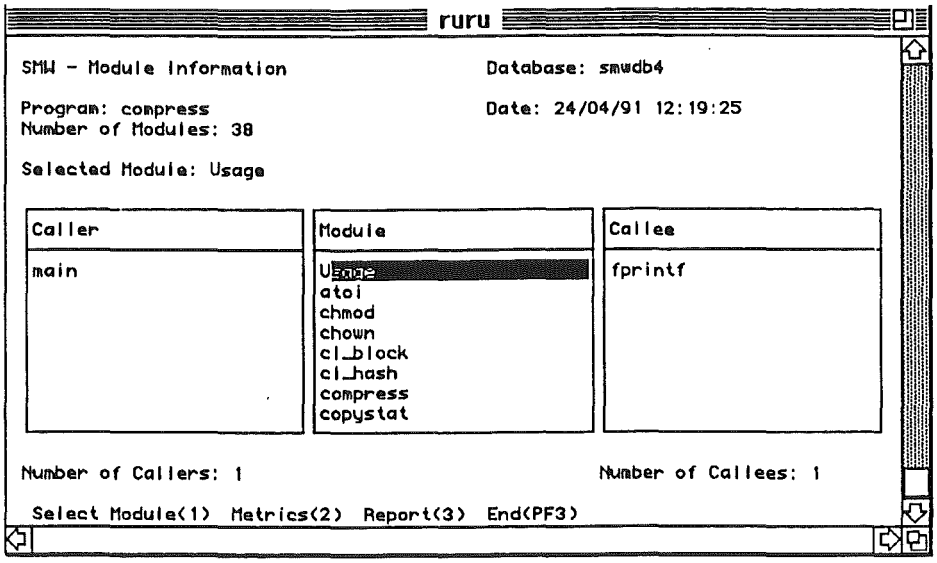


Figure 3.9 SMW Module Information Screen.

There are two reports available.

The All Modules report consists of a sorted list of all the modules in the program version, with information on what modules call, and are called by the module. This report file will have the format described in Appendix A, Section A5.

The Selected Module report is a list of what modules a selected module calls and is called by. This report file will have the format described in Appendix A, Section A6.

The procedure for generating the report, All Module, will be dealt with first, with the procedure for the Selected Module report second.

All Modules

1. At the SMW Program Version Information screen move the cursor to the version you want to generate a report for.

2. Select from the SMW Program Version Information Screen menu the menu option

   `Structure`

   This will display the SMW Module Information screen.

3. Bring up the report sub-menu (Figure 3.10) by selecting the menu option

   `Report`

4. Select the menu option

   `All Modules`

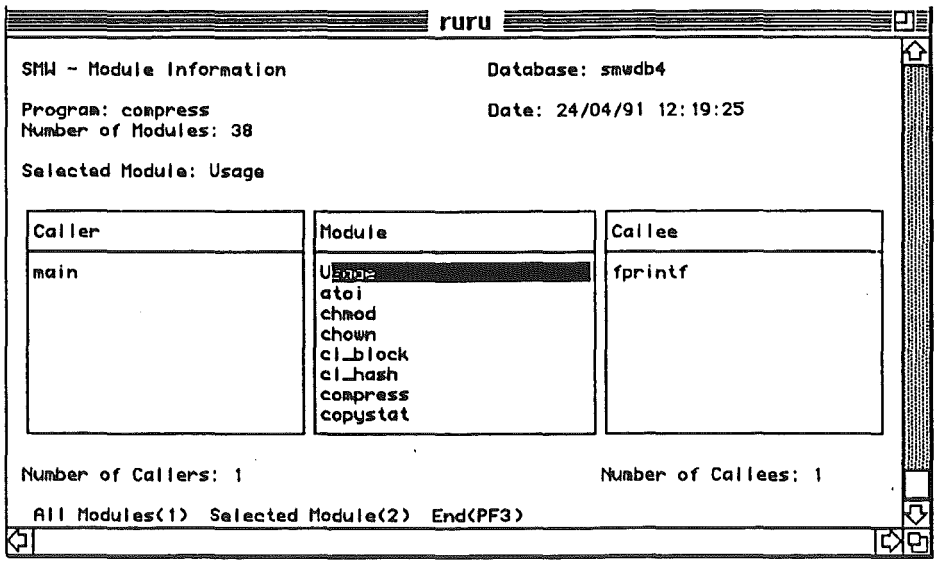


Figure 3.10 SMW Module Information Screen – Report Sub-menu.

5. When prompted the user should enter the name of the file to output the report to and press return. If no file name is entered then no report will generated and the user will be returned to the sub-menu where the report type was selected from.

6. When the report has been created the user will be returned to the sub-menu where the report type was selected from and can then select another (or the same) report option or can select

   `End`

   and return to the SMW Module Information screen menu.

Selected Module

1. At the SMW Program Version Information screen move the cursor to the version you want to generate a report for.

2. Select from the SMW Program Version Information Screen menu the menu option

   `Structure`

   This will display the SMW Module Information screen.

3. Move the cursor until it is on the name of the module you wish to have generate the report on.

   `Select Module`

4. Bring up the report sub-menu (Figure 3.10) by selecting the menu option

   `Report`

5. Select the menu option

   `Selected Module`

6. When prompted the user should enter the name of the file to output the report to and press return. If no file name is entered then no report will generated and the user will be returned to the sub-menu where the report type was selected from.

7. When the report has been created the user will be returned to the sub-menu where the report type was selected from and can then select another (or the same) report option or can select

   `End`

   and return to the SMW Module Information screen menu.

3.5.2 Plots

The SMW system provides the facility for generating plots of metrics. This feature is available from the SMW Program Version Information screen and allows two different metrics from a program version to be plotted against each other. This facility uses the GNUPlot program, and the plot is output as a PostScript file suitable for downloading to a PostScript laserprinter or a PostScript previewing program.

The procedure for generating a plot is as follows.

1. At the SMW Program Version Information screen move the cursor to the program version you want to select.

2. Bring up the SMW Metric Data Plotting screen (Figure 3.11) by selecting the menu option

   `Plot`

3. Select the metric to be plotted on the X-axis by moving the cursor to the metric name and selecting

   `X Select`

from the SMW Metric Data Plotting screen menu.

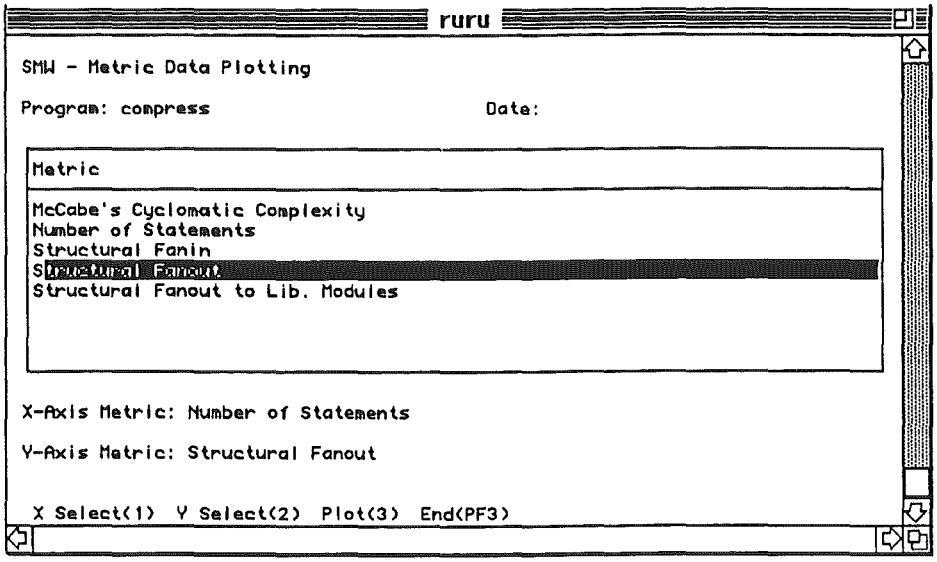


Figure 3.11 SMW Metric Data Plotting Screen

4. Select the metric to be plotted on the Y-axis by moving the cursor to the metric name and selecting

   `Y Select`

   from the SMW Metric Data Plotting screen menu.

5. Plot the data by selecting the menu option

   `Plot`

   from the SMW Metric Data Plotting screen menu.

6. When prompted the user should enter the name of the file to output the plot to and press return. If no file name is entered then no plot will generated and the user will be returned to the SMW Metric Data Plotting screen menu.

7. When the plot has been created the user will be returned to the SMW Metric Data Plotting screen menu and can then select another plot or can select

   `End`

   and return to the SMW Program Version Information screen menu.

3.5.3 Interactive queries.

The user can perform a range of interactive queries on an SMW database. These can range from simple queries specified from a menu, to more elaborate queries available through

various Ingres sub-systems such as QBF and ISQL. The latter provide the user with more power and flexibility, but require the user to be knowledgable about the internal structure of an SMW database. The types of queries that SMW supports will now be discussed in the following sections.

Simple queries

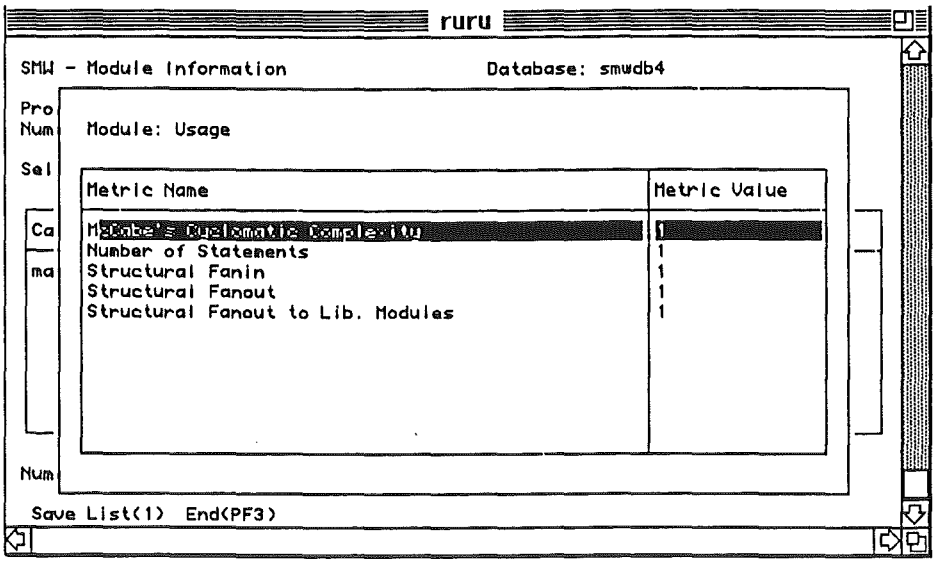


Figure 3.12 SMW Module Information Screen – Metrics sub-menu

Simple queries in the SMW system involve simply selecting a menu option and the system returning the data requested. Often this data will be presented in a box on the screen and a sub-menu will appear giving the user the option of saving the data to a file (Figure 3.12.). Simple queries are available on the SMW Program Version Information screen (Metrics menu option) and the SMW Module Information screen (Select Module and Metrics menu options). For more information on these menu options see Section 4.4.

SMW query forms.

At present there is only one SMW query form present in the system. A query form is a screen that allows the user to enter in conditions on the data to retrieve. The SMW Query System Form can be accessed in the following way.

1. At the SMW Program Version Information screen move the cursor to the program version you want to query..

2. Bring up the SMW Query System form (Figure 3.13) by selecting the menu option

   `Query`

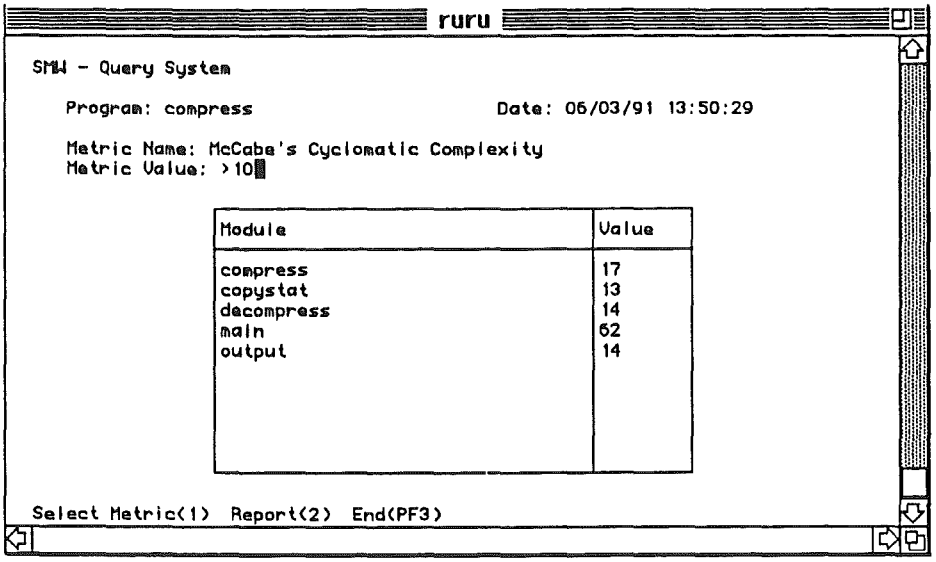


Figure 3.13 SMW Query System form.

3. Enter the condition for the metric value in the field

   `Metric Value:`

4. Choose the menu option

   `Select Metric`

5. Move the cursor to the name of the metric data to retrieve and select the menu option

   `Select`

   If the user selects

   `End`

   instead at this point then nothing will happen and the user will be returned to the SMW Query Form screen menu.

6. If you want to save the data select the menu option

   `Report`

   and when prompted the user should enter the name of the file to output the data to and press return. If no file name is entered and return pressed nothing will be saved.

7. When finished the user selects the menu option

   `End`

   and returns to the SMW Program Version Information screen menu.

Complex queries.

Complex queries a possible through the use of a variety of Ingres sub-systems. The sub-systems currently supported are Query-By-Forms (QBF), the IQUEL (Interactive QUEL) monitor and the ISQL (Interactive SQL) monitor. The use of these requires the user to have a good understanding of the internal structure of an SMW database, as well as being familiar with the query tools. The SMW database can also be queried through the use of embedded query languages in user written programs.

The following procedure describes how to access the Ingres sub-systems supported by the SMW menu interface.

1. Select from the SMW Program Information menu the menu option

   `Query`

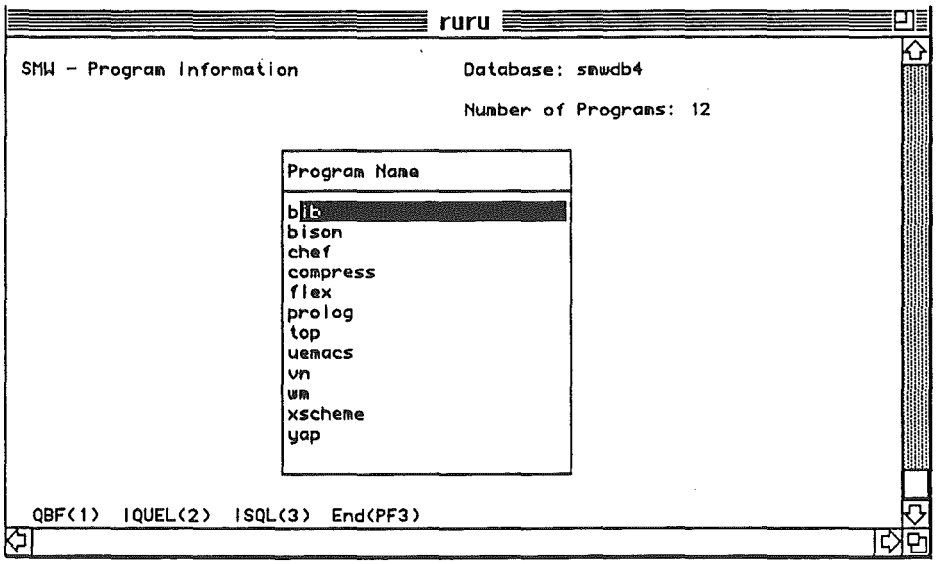


Figure 3.14 SMW Program Information Screen – Query Sub-System Sub-menu.

2. Select from the sub-menu presented (Figure 3.14) one of

   `QBF`

   or

   `IQUEL`

   or

   `ISQL`

3. After exiting from the sub-system select

   `End`

   to return to the Program Information screen menu.

4. Guide to the SMW Screens

4.1 SMW Title Screen

Purpose.

The purpose of the SMW Title screen is to provide the user with the facility to create new SMW databases, destroy old SMW databases, and to select an existing SMW database for browsing.

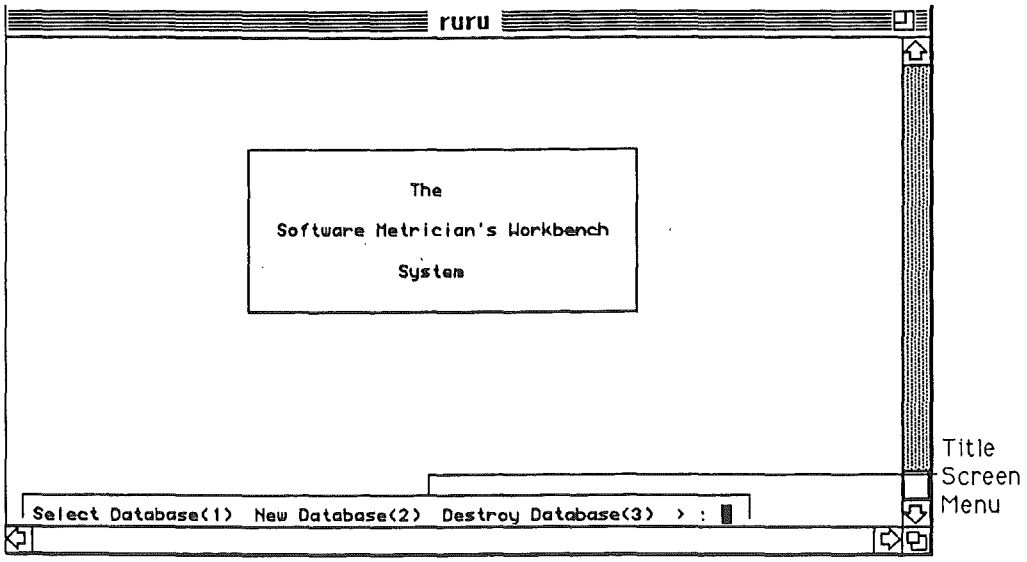


Figure 4.1. - SMW Title Screen

Menu Options

`Select Database` – Allows the user to select an existing database to browse. Upon selecting this option the user will be prompted to enter the name of the database to browse.

`New Database` – Allows the user to create a new SMW database. Upon selecting this option the user will be prompted to enter the name of the database to create.

`Destroy Database` – Allows the user to destroy an existing database. Upon selecting this option the user will be prompted to enter the name of the database to destroy.

`Quit` – This option causes the user to exit form the SMW Browser program.

4.2 SMW Program Information Screen

Purpose.

The purpose of the SMW Program Information screen is to provide the user with a view of what programs exist in the selected SMW database.

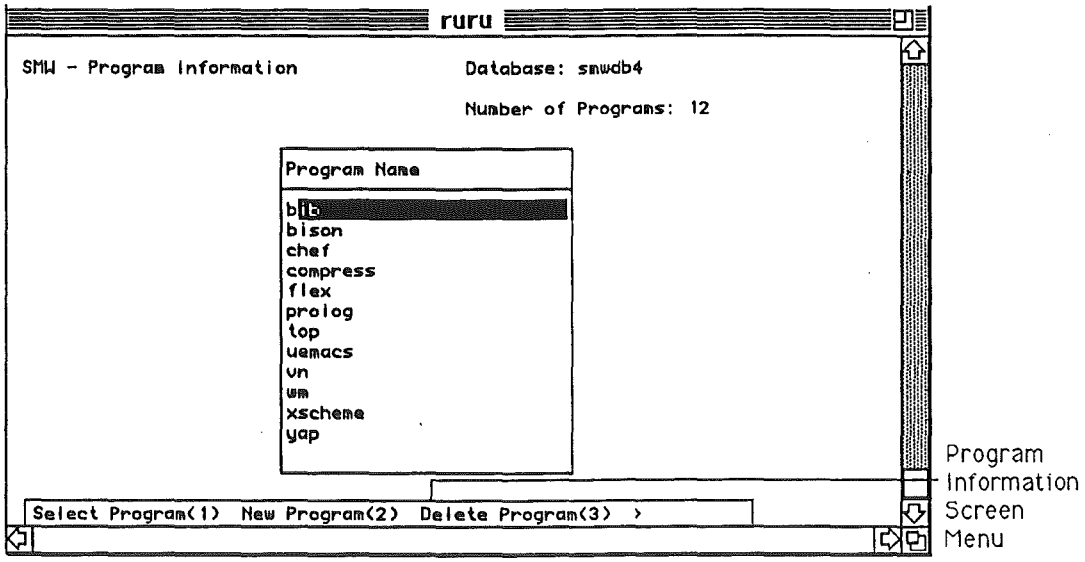


Figure 4.2. - SMW Program Information Screen.

Menu Options

`Select Program` – When this option is selected the SMW Program Version Information screen will be displayed for the program that the cursor was sitting on.

`New Program` – This option allows the user to collect basic program information for a new program or program version. Upon selection the following sub-menu options are displayed.

`Metric Shell` – Selecting this option will take the user into the Metrics Shell program.

`UNIX Shell` – Selecting this option will take the user into an interactive UNIX shell, in order to run programs from the command line. (See Section 5.2.3).

`End` – This option causes the user to leave this sub-menu and return to the previous menu.

Delete Program – This option deletes all the information about any versions of the program that has the cursor on it's name form the selected SMW database. The user is prompted to confirm whether of not the program is to be deleted.

`Rename Program` – Selecting this option allows the user to rename a program and it's versions.

`Query` – Selecting this option brings up the following sub-menu showing the various Ingres sub-systems available for ad-hoc queries. (See the Ingres manuals for further details.)

`QBF` – Query-By-Forms. (Forms can be created by the Ingres System Administrator).

`IQUEL` - Interactive QUEL monitor.

`ISQL` - Interactive SQL monitor.

`End` – This option causes the user to leave this sub-menu and return to the previous menu.

`Report` – This option brings up the following sub-menu allowing the user to generate reports on the information displayed on this screen.

`Only Program Names` – This option generates a report containing a list of the program names in the database and puts it into the file specified by the user. See Appendix A, Section A1 for the format of the report.

`Program Names and Versions` – This option generates a report containing a list of the program names in the database and their associated program versions and puts it into the file specified by the user. See Appendix A, Section A2 for the format of the report.

`End` – This option causes the user to leave this sub-menu and return to the previous menu.

End – This option takes the user back to the SMW Title screen.

4.3 SMW Program Version Information Screen

Purpose.

The purpose of the SMW Program Version screen is to display the versions of the program the user selected in the SMW Program Information screen to view information about a version, such as metric information, and to add to and report on this information.

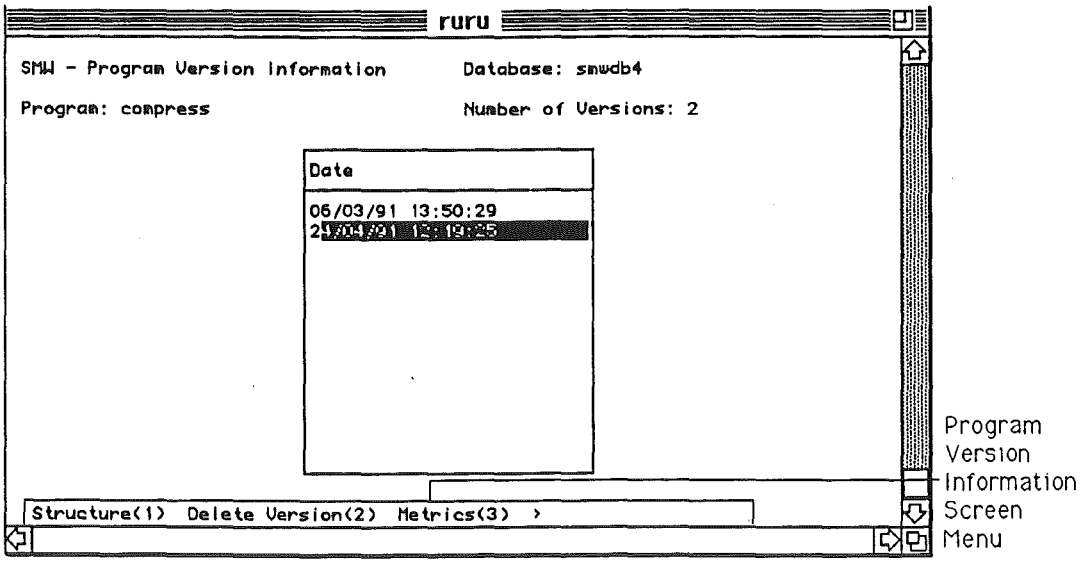


Figure 4.3. - SMW Program Version Screen

Menu Option

`Structure` – This menu option takes the user to the SMW Module Information screen so that the user can examine the structure of the program version the cursor was on.

`Delete Version` – Deletes all information on the program version the cursor is on from the SMW database being browsed.

`Metrics` – Displays a list of what metrics have been calculated for the program version the cursor is on. The following sub-menu is also displayed.

`Save` – Allows the user to save the list of metrics being displayed to a file.

`End` – This option causes the user to leave this sub-menu and return to the previous menu.

`Calculate Metric` – Selecting this option brings up a list of metrics and displays a sub-menu allowing the user to calculate new metric values for the program version the cursor was on. The sub-menu is as follows:

`Select` – Selects the metric the cursor is on and if that metric hasn't already been calculated for this version, calculates the metric values and puts them into the database.

`End` – This option causes the user to leave this sub-menu and return to the previous menu.

Plot – Selecting this option takes the user to the SMW Metric Data Plotting screen to plot data for the program version the cursor was on.

Report – Selecting this menu option brings up the following sub-menu which allows the user to generate reports on the program version the cursor is on.

`All Metrics` – Generates a report on all the metrics calculated for the selected program version and outputs the report into a file specified by the user. See Appendix A, Section A3 for the format of the report.

`Selected Metric` – Selecting this menu option causes a list of metric names that have values calculated for this program version to be displayed and the following sub-menu to be displayed. See Appendix A, Section A4 for the format of the report.

`Select` - This menu option will cause a report to be generated for the selected program version consisting on the metric values of the metric name that the cursor was on.

`End` – This option causes the user to leave this sub-menu and return to the previous menu

`End` – This option causes the user to leave this sub-menu and return to the previous menu.

Query – This menu option takes the user to the SMW Query System screen for querying the program version that had the cursor on it.

End – This option takes the user back to the SMW Program Information screen.

4.4 SMW Module Information Screen

Purpose.

The purpose of the SMW Module Information screen is to provide the user with information about the structure of a program version at a module level.

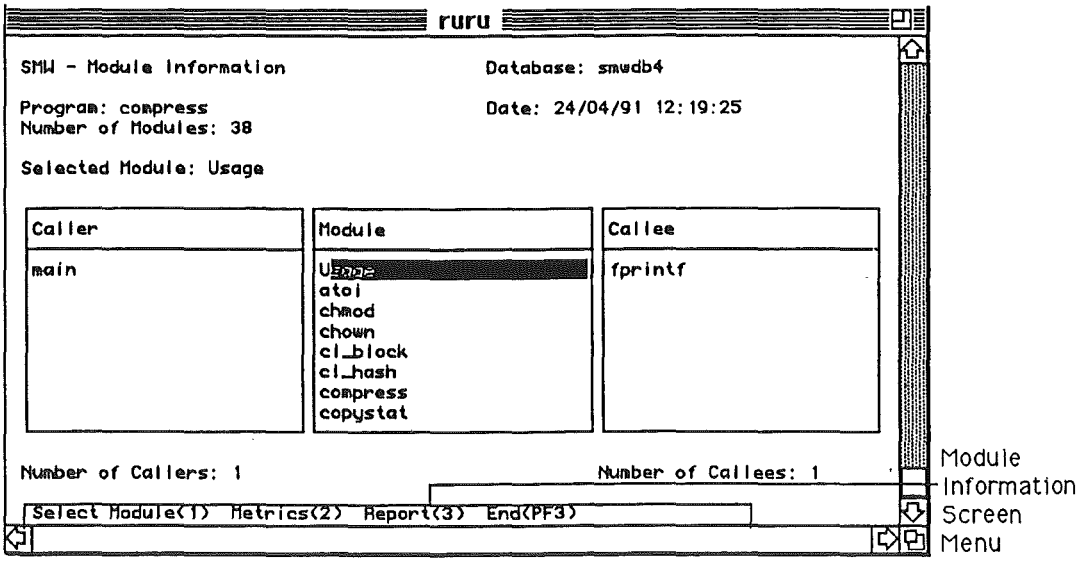


Figure 4.4. - SMW Module Information Screen.

Menu Options.

`Select Module` – This option causes information on the module the cursor is currently on to be retrieved from the database and displayed. The modules which call the selected module are displayed in the "Callers" list and the modules called by the selected module are displayed in the "Callees" list.

`Metrics` – This option causes a list to be displayed showing the metric values calculated for the module the cursor was on. The following sub-menu is also displayed.

`Save` – Allows the user to save the list of metrics being displayed to a file.

`End` – This option causes the user to leave this sub-menu and return to the previous menu.

`Report` – This menu option allows the use to generate reports on the program version structure. The following sub-menu will be displayed.

`All Modules` – A report will be generated for all the modules in the program version containing information on what modules call and are called by each module. See Appendix A, Section A5 for the format of the report.

`Selected Module` - This option causes a report to be generated on what modules are called by and call the module the cursor is currently on. See Appendix A, Section A6 for the format of the report.

`End` – This option causes the user to leave this sub-menu and return to the previous menu.

`End` – This menu option returns the user to the SMW Program Version Information screen.

4.5 SMW Metric Data Plotting Screen

Purpose.

The purpose of the SMW Metric Data Plotting screen is to provide the user with a way of graphically comparing two metrics calculated for the same program version.

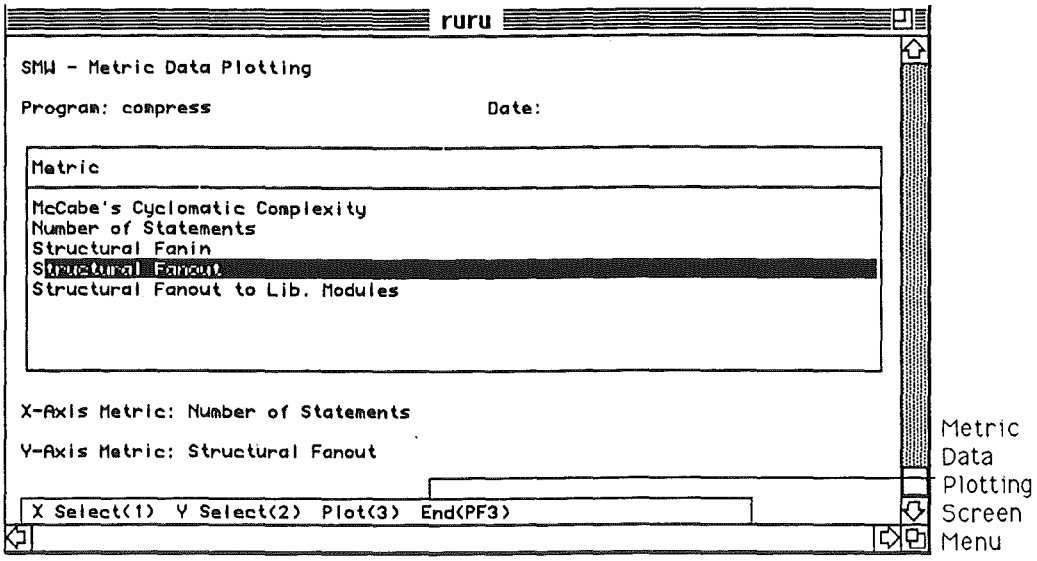


Figure 4.5. - SMW Metric Data Plotting Screen.

Menu Options.

`X Select` – This option selects the metric the cursor is on to be the metric whose values will be used as the X coordinate.

`Y Select` – This option selects the metric the cursor is on to be the metric whose values will be used as the Y coordinate.

`Plot` – Outputs the plot as a PostScript file with a name of the user's choice. (See the System Administrator for information on how to print/display PostScript files).

`End` – This option causes the user to leave this sub-menu and return to the previous menu.

4.6 SMW Query System Screen

Purpose.

The purpose of this screen is to provide the user with a way of displaying selected values of a specified metric for a program version.

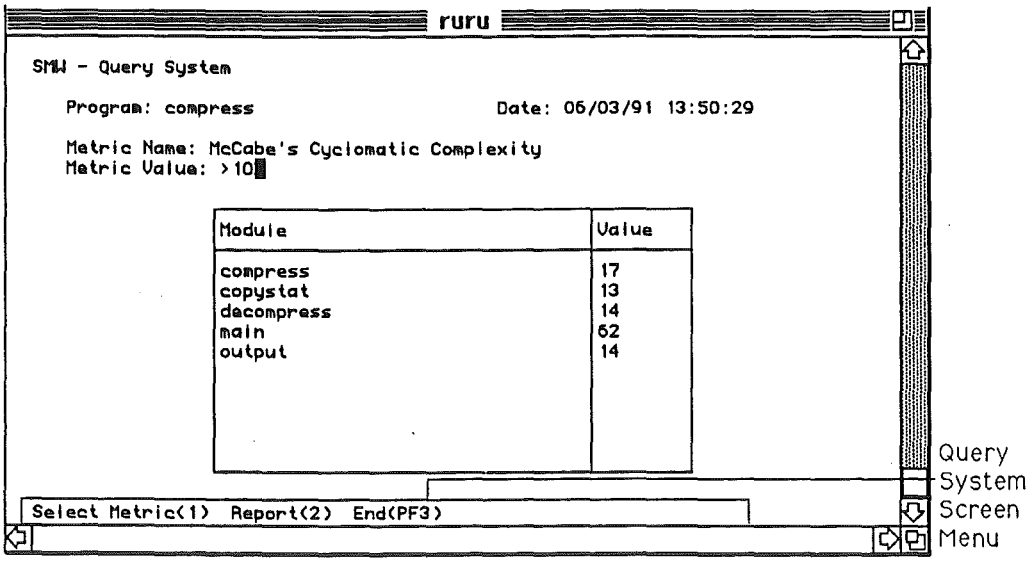


Figure 4.6. - SMW Query System Screen

Menu Options.

`Select Metric` – This option is selected after the user has entered a condition in the "Metric Value" field of the screen. After selecting this option a list of metrics will be displayed and the following sub-menu will be displayed.

> `Select` – This option selects the metric name the cursor is on in the list of metrics as the metric to retrieve the information for and display on the screen.
>
> `End` – This option causes the user to leave this sub-menu and return to the previous menu.

`Report` – This menu option allows the user to output the information on the screen into a file of the user's choice.

`End` – This option returns the user to the SMW Program Version Information screen.

4.7 SMW Metrics Shell Screen

Purpose.

The purpose of this screen is to provide the user with a way of collecting basic program information about a program by graphically selecting the files to select the information from on the screen, saving the user from having to deal with a command line.

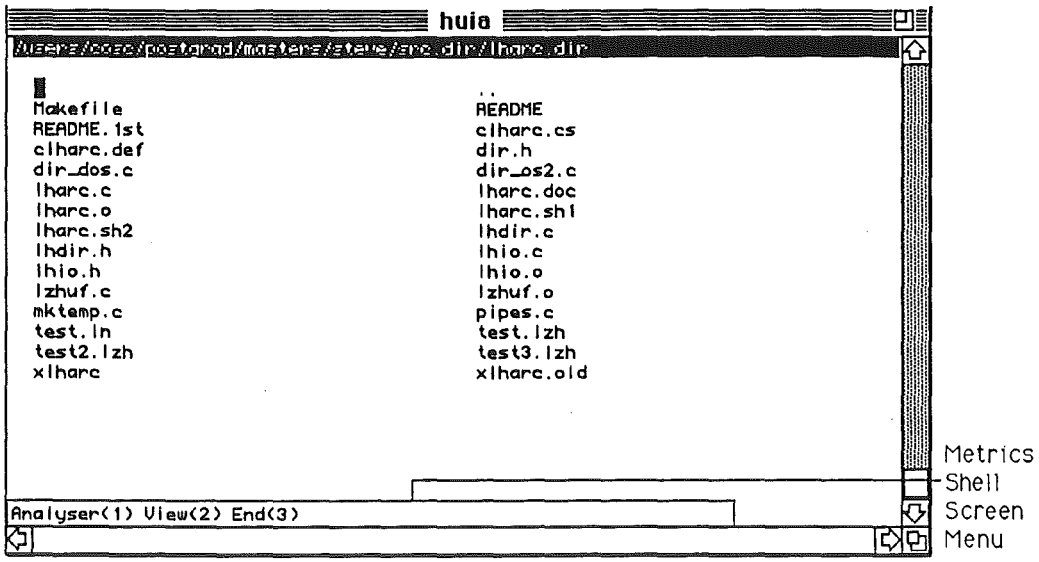


Figure 4.7. - SMW Metrics Shell Screen

Keys

The following tables show what functions the various keys perform.

Movement	Keys
Next Row	j, down arrow, return
Previous Row	k, up arrow
Next Column	l, right arrow, tab
Previous Column	h, left arrow

Table 4.1 Movement Keys

Other	Key
Analyser	1
View	2
End	3
Tag/Untag File	space
Change Directory	c

Table 4.2 Other Keys

To change directories, move the cursor onto the name of the directory (or link) in the list of files on the screen and press the 'c' key. If the name the cursor is on is a file then nothing will happen.

To tag a file, move the cursor onto the name of the file and press the space key. The file should be highlighted and an asterisk should appear beside it on the left. To remove the tag, move to the tagged file and press the space key. If the file name is a directory then the entry will not be tagged.

Menu Options.

`Analyser` – This option causes the C-Flow Analyser to be run on the files that have been tagged on the screen. If no files have been tagged the file that the cursor is on will have the analyser run on it. After the analyser has run the user will be returned to the SMW Program Information screen.

`View` – This option allows the user to view text files. All the tagged files will be viewed one after another. If no files are tagged then the file the cursor is on will be viewed if possible.

`End` – This option returns the user to the SMW Program Information screen.

5. The Programs That Make Up the SMW System.

5.1 Introduction.

The SMW System is a collection of programs that complement each other. If the user is using the SMW forms interface then this collection of programs is hidden from the users view, but sometimes there are times when the user will want to access the individual programs. For example, if the user wants to run the collection of metrics analysis programs in a batch run overnight then it is not feasible to do this from the SMW interface. Also a selection of tools for retrieving information from a database, in order to do ad-hoc queries or database administration.

These programs will be discussed in accordance with the tasks that they perform and a brief description of their purposes and usage is given for each.

5.2 The programs and their use.

The programs in the SMW system can be divided up into the following categories:

- User interface - the programs that provide the user's forms interface.
- Metrics Collection - programs that calculate metric data for a program
- Program Data Collection - programs that collect the basic program information
- System Administration - programs that aid the user in maintaining the system and using it outside of the forms interface.

All the SMW programs are located in one directory the location of which should be specified in the environment variable *SMWPATH*.

5.2.1 User interface programs.

There are two user interface programs in the SMW system. One is the SMW Browser, and the other is the Metrics Shell.

The SMW Browser.

The SMW Browser is the program the user sees when he or she starts up the SMW system and is presented with the forms system interface. It provides a relatively easy way for the user to access the capabilities of the SMW system without having to know about the

structure of the database.

The SMW Browser program takes no command line arguments and can be run from the shell prompt by simply typing

```
smw
```

The Metrics Shell.

The Metrics Shell program is designed to give the user a display of the current directory and the ability to perform various operations on the files and directories in the directory. It's primary purpose is to allow the user to select the source files for the basic program information to be extracted from and to allow the user to change directories and to view files.

The Metrics Shell is called by the SMW Browser program when the user selects the Metrics Shell menu option from the New Program sub-menu. When the Metrics Shell has collected the basic program information for a program, it exits back to the user's shell program.

To run the Metrics shell from the command line type

```
metsh <database> <program> <directory path>
```

where

<database> is the name of the database to add the new program to.

<program> is the name of the new program to be added.

<directory path> is the full path name describing the location of the source files.

5.2.2 Program data collection programs.

The is currently one program to support the collection of basic program information, the C-Flow Analyser. This program takes the output from the C-Flow program as it's input and uses that information to build up a basic picture of the program's structure described in the input. The C-Flow Analyser is called from the Metrics Shell program and can be used from the command line as follows:

```
cfa <database> <program> <directory path> < <input file>
```

where

<database> is the name of the database to add the new program to.

<program> is the name of the new program to be added.

<directory path> is the full path name describing the location of the source files.

<input file> is the file generated by running the C-Flow program on the C source files to capture the basic program information from.

For example:

```
cflow file1.c file2.c file3.c > files.cflow
cfa smwdb prog1 /usr/users/masters/steve < files.cflow
```

A description of the C-Flow program can be found in Section 1 of the on-line UNIX manual pages.

5.2.3 Metrics collection programs.

The metrics collection programs are a set of programs that each calculate a metric or set of metrics for a program that has been entered in the database. Some programs use data already in the database to calculate the metrics (e.g. fanin-fanout), while others take input from outside the database and use that. All programs store the information calculated in the database.

Fanin-fanout

This program calculates the fanin and fanout values for a program whose basic program information has already been entered in the database. The program calculates the fanin values for each module in the program version, the fanout value for each module that is not a library module, and the fanout value to library modules from modules that are not library modules. This program is called from the SMW Browser, but can be run from the command line as follows:

```
fanin-fanout <database> <program ID>
```

where

<database > is the name of the database where the program version information is stored.

<program ID> is the integer identifier that uniquely identifies the program version in the specified database

McCabe's Cyclomatic Complexity, NPATH, and Number of Statements programs.

These three programs calculate the metric values for cyclomatic complexity, NPATH and the number of statements for each function in a C source file. The results are then put into the SMW database. The C source file must be run through the C preprocessor first before being input into the metrics program.

These programs are called from the SMW Browser, but can be run from the command line as follows:

McCabe's cyclomatic complexity

```
mccabe <database> <program ID> < <C source file>
```

NPATH

```
npath <database> <program ID> < <C source file>
```

Number of statements

```
proginfo <database> <program ID> < <C source file>
```

where

<database> is the name of the database where the program version information is stored.

<program ID> is the integer identifier that uniquely identifies the program version in the specified database

The best way to collect the metric values for a group of program source files is probably to use a shell script. For example in the C Shell:

```
foreach f (file1.c file2.c file3.c)
cc -E $f | mccabe smwdb 16
end
```

5.2.4 System administration programs.

The SMW system currently provides four tools for the use of the administrator of an SMW database. The first two tools provide information on the programs and metrics in the database, the second tool provides a way of adding metrics to an SMW database, and the two other tools deal with the deletion of metric information from a database.

SMW-Programs Program.

This program provides the administrator with information on all the programs in an SMW database. The program produces a report containing information on each program. (See Appendix A7 for the format of the report).

To run the program type the following on the command line:

```
smw-programs <database>
```

where

<database> is the name of the SMW database.

SMW-Metrics Program.

This program provides the administrator with information on all the metrics in an SMW database. The program produces a report containing information on each metric. (See Appendix A8 for the format of the report).

To run the program type the following on the command line:

```
smw-metrics <database>
```

where

<database> is the name of the SMW database.

Add-Metric Program.

Before a metric can be calculated for a program the SMW system must have some basic metric information input into the database. This information includes the name of the software metric, and the name of the program that generates that metric information. To add this information to the database enter the following on the command line:

```
add-metric <database> <metric name> <metric program>
```

where

<database> is the name of the database to add the metric information to.

<metric name> is the name of the new metric to be added.1

<metric program> is the name of the metric program that generates the metric values.

Deleting Metric Information

There are two programs available for deleting metric information form an SMW database. Firstly, there is a program that deletes all metric values of a particular metric that have been calculated for a specific program version. Secondly, there is a program that destroys all information about a particular metric in the database, including metric values, the metric name and the name of the program that generates the metrics.

Delete-Program-Metric

The program deletes all the metric values for a particular metric for a specific program version in the database. The program has the following command line form

```
delete-program-metric <database> <program ID> <metric name>
```

where

<database> is the name of the database to delete the metric information from.

<program ID> is the identifier that uniquely identifies the version of the

---

[1]If the metric name is more than one word then the words that make up the name should be enclosed in double quotes. For example: "McCabe's Cyclomatic Complexity"

program.[2]

<metric name> is the name of the metric to delete the metric values of.3

Destroy-Metric-Info

This program destroys all the information about a metric in the specified database. All program versions that have values calculated for this metric will have them destroyed and all the information concerning the metric names and programs will also be destroyed. To use the metric program again to calculate metric values for the database will then require the metric to be re-added to the database using the `add-metric` program. To destroy all the metric information in a database type the following on the command line.

destroy-metric-info <database> <metric name>

where

<database> is the name of the database.

<metric name> is the name of the metric.[4]

5.2.5 Summary of SMW System Administration Programs.

- `smw-programs` - generates report of all program versions in database.
- `smw-metrics` - generates report of all metrics in database.
- `add-metric` - adds new metric to database.
- `delete-program-metric` - deletes all metric values for a program version for a specified metric.
- `destroy-metric-info` - destroys all information on specified metric from database.

---

[2]This information can be obtained using the `smw-programs` program.

[3]If the metric name is more than one word then the words that make up the name should be enclosed in double quotes. For example: `"McCabe's Cyclomatic Complexity"`

[4]If the metric name is more than one word then the words that make up the name should be enclosed in double quotes. For example: `"McCabe's Cyclomatic Complexity"`

Appendix A. SMW Report Formats

A1. Program Information report - Only Program Names

```
Programs in Database:<tab><database name>

<program_1 name>
<program_2 name>
...
<program_n name>
```

A2. Program Information report - Program Names and Version

```
Programs in Database:<tab><database name>

<program_1 name>
<tab><program_1 version_1>
<tab><program_1 version_2>
<program_2 name>
...
<program_n name>
<tab><program_n version_1>
<tab><program_n version_2>
```

A3. Program Version Information report - All Metrics

```
Program:<tab><program name>
Date:<tab><version name>

Module<tab><metric_1 name><tab>...<metric_n name>
<module_1 name><tab><metric_1 value><tab>...<metric_n value>
<module_2 name><tab><metric_1 value><tab>...<metric_n value>
...
<module_n name><tab><metric_1 value><tab>...<metric_n value>
```

A4. Program Version Information report - Selected Metric

```
Program:<tab><program name>
Date:<tab><version name>

Metric:<tab><metric name>

Module Name<tab>Value
<module_1 name><tab><metric value>
<module_2 name><tab><metric value>
...
<module_n name><tab><metric value>
```

A5. Structure Information report - All Modules

```
Program:<tab><program name>
Program Date:<tab><version name>

Module:<tab><module_1 name>

<tab><tab>Calls:
<tab><tab><tab><module name>
...
<tab><tab>Called by:
<tab><tab><tab><module name>
...
```

A6. Structure Module Info - Selected Module

```
Program:<tab><program name>
Program Date:<tab><version name>

Module:<tab><module name>

<tab><tab>Calls:
<tab><tab><tab><module name>
...
<tab><tab><tab><module name>
<tab><tab>Called by:
<tab><tab><tab><module name>
...
<tab><tab><tab><module name>
```

A7. smw-programs report.

```
Name:<tab><program name>
Date:<tab><version date>
ID:<tab><program ID>
Location:<tab><program directory>
Files:
<tab><program file1>
<tab><program file2>
```

where

<program name> is the name of the program.

<version date> is the date this version of the program was entered into the database.

<program ID> is the identifier that uniquely identifies this version of the program.

<program directory> is the full path name of the source directory.

<program $file_n$> is one of the files that contains part of this program version.

A8. smw-metrics report

```
Name:<tab><metric name>
ID:<tab><metric ID>
Program:<tab><metric program>
```

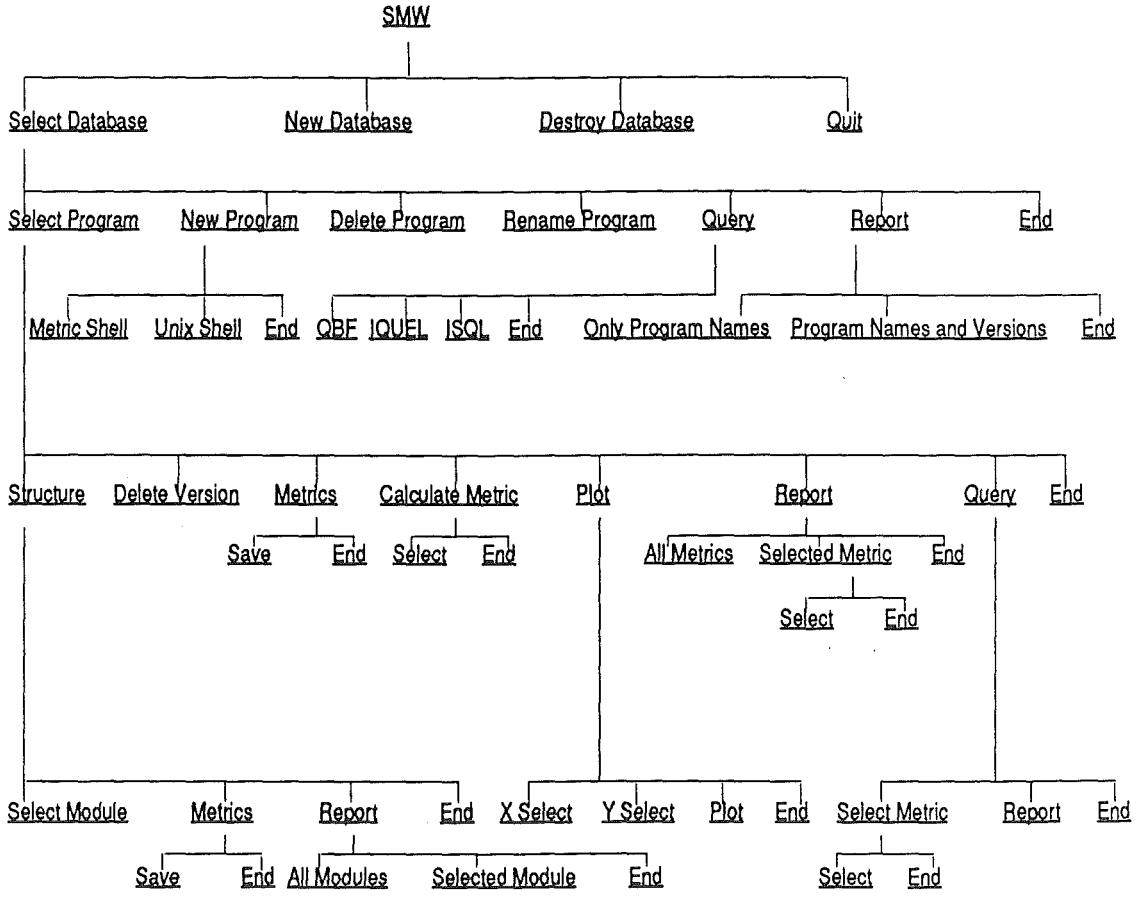
where

`<metric name>` is the name of the metric.

`<metric ID>` is the identifier that uniquely identifies the metric.

`<metric program>` is the name of the program that generates the metric values.

Appendix B. SMW Forms Interface Command Structure



Appendix C. The NPATH Software Product Metric

A metric that counts the number of execution paths through functions written in the C programming language[1].

C.1 Background Definitions

A control flow graph is a graph in which each vertex represents either a basic block of code (statement sequence that contains no branches) or a branch point in the function, and each edge represents a possible flow of control. More formally the control flow graph of a function can be represented as a directed graph four–tuple ($V, E, V_{entry}, V_{exit}$), where

• V is a set of vertices representing basic blocks or branch points in the function;

• E is a set of edges representing flow of control in the function;

• V_{entry}, an element of V, is the unique function entry vertex; and

• V_{exit}, an element of V, is the unique function exit vertex.

• A possible execution path is any path from V_{entry} to V_{exit} in a flow graph.

• An elementary cycle is any path P from vertex $V_1, V_2, \ldots, V_k$ such that $V_1 = V_k$ and $V_i \neq V_j$ for $1 < i < j <= k$. In short, an elementary cycle is a cycle that contains no other cycles within it.

• A loop control vertex is a vertex V with the following two properties: (1) V has an out-edge that lies on at least one elementary cycle that begins and ends at V, and (2) V has a second out-edge that lies on a path leading out of the loop.

• A loop is a cycle that begins and ends at a given loop control vertex.

• A range of a statement V is the set of statements whose execution may be determined by the truth value of the expression in statement V.

---

[1]Nejmeh, B A. NPATH: A Measure of Execution Path Complexity. 1988; Comm. ACM; Vol 31; No 2; pp 188-200

The Execution Path Complexity of C Functions

The composite acyclic execution path complexity for a C function, NPATH, is

$$NPATH = \prod NP(Statement_i)$$

for i = 1 to N, where N is the number of statements in the function and $NP(Statement_i)$ is the acyclic execution path of $statement_i$.

C.2 NPATH Execution Path Complexity of C Control Flow Structures

The following section defines the acyclic execution path expression complexities (NP) for the control flow structures in the C programming language.

if Statement

Syntax

```
if (<expr>)
     <if-range>
S;
```

```
NP(if)    =    NP(<if-range>) + NP(<expr>) + 1
```

if–else Statement

Syntax

```
if (<expr>)
     <if-range>
else
     <else-range>
S;
```

```
NP(if-else)    = NP(<if-range>) + NP(<else-range>) +
                 NP(<expr>)
```

while Statement

Syntax

```
while (<expr>)
     <while-range>
S;
```

NP(while) = NP(<while-range>) + NP(<expr>) + 1

do while Statement

Syntax

```
do
     <do-range>
while (<expr>)
S;
```

NP(do) = NP(<do-range>) + NP(<expr>) + 1

for Statement

Syntax

```
for (<expr1>; <expr2>; (<expr3>)
     <for-range>
S;
```

NP(for) = NP(<for-range>) + NP(<expr$_1$>) + NP(<expr$_2$>) + NP(<expr$_3$>) + 1

switch Statement

Syntax

```
switch (<expr>)
     {
          <case-range_1>
          .
          .
          .
          <case-range_n>
          <default-range>
     }
```

$NP(switch) = NP(<expr>) + NP(<default\text{-}range>) + \sum NP(<case\text{-}range_i>)$

? Operator

Syntax

$<expr_1>\ ?\ <expr_2>\ :\ <expr_3>$

$NP(?) = NP(<expr_1>) + NP(<expr_2>) + NP(<expr_3>) + 2$

---

goto Statement

Syntax

```
goto <label>
```

The `goto` statement is not accounted for by this metric.

---

break Statement

Syntax

`break`

`NP(break)` = 1

Expressions

Syntax

$\texttt{<expr}_1\texttt{>}\ \texttt{op}_1\ \texttt{<expr}_2\texttt{>}\ \texttt{op}_2\ \ldots\ \texttt{op}_{N-1}\ \texttt{<expr}_N\texttt{>}$

where $\texttt{op}_1$, $\texttt{op}_2$, ..., $\texttt{op}_{N-1}$ are any one of the logical operators and (&&) or or (||)

`NP(expression)` = number of && and || operators in the expression

---

continue Statement

Syntax

`continue`

The `continue` Statement is not accounted for by this metric.

---

return Statement

Syntax

$\texttt{return <expr}_{opt}\texttt{>}$

The `return` statement can have an optional expression attached to it.

$\texttt{NP(return)} = \texttt{NP(<expr}_{opt}\texttt{>)}$

Sequential Statements and Function Calls

The execution path complexity for the sequential statement is 1 because there is only one path created by consecutive sequential statements. Function calls are treated as sequential statements.

```
NP(sequential) = 1
```

Appendix D. SMW Test Program Data Set.

The SMW test program data set includes twelve programs of various sizes and applications written in source code able to be processed by the cflow/cflow analyser tools.

This appendix is divided in sections covering different aspects of the program and metric data from the program data set. These sections are Appendices D1 through to and including D5, and are divided into the following topics.

Appendix D1 - Program Information for each program in the test set.

Appendix D2 - Metric values calculated for each program in the test set.

Appendix D3 - Basic statistics for the sets of metric values for each program in the test set.

Appendix D4 - Linear correlations between the sets of metric values for each program in the test set.

Appendix D5 - Frequency distributions for the sets of metric values in total.

Appendix D1 - Program Information.

Program:	**bib**		
Description:	Program for collecting and formatting reference lists in		documents.
Version:	2.9		
Date:	23/7/85		
Total No. Modules:	93		
No. Non-Library Modules:	62		
No. Library Modules:	31		
Files:	alpha.seek.c	bib.c	bibargs.c
	locate.c	makekey.c	streams.c

Program:	**bison**		
Description:	GNU Project Parser Generator.		
Total No. Modules.	150		
No. Non-Library Modules:	129		
No. Library Modules:	21		
Files:	LR0.c	allocate.c	closure.c
	conflicts.c	derives.c	files.c
	getargs.c	lalr.c	lex.c
	main.c	nullable.c	output.c
	print.c	reader.c	symtab.c
	warshall.c		

Program:	**chef**		
Description:	Command and display oriented text editor.		
Date:	7/12/87		
Total No. Modules:	227		
No. Non-Library Modules:	190		
No. Library Modules:	37		
Files:	chef1.c	chef10.c	chef2.c
	chef3.c	chef4.c	chef4a.c
	chef5.c	chef6.c	chef7.c
	chef7a.c	chef8.c	chef9.c

Program:	**compress**		
Description:	UNIX file compression utility.		
Version:	2.0		
Total No. Modules:	38		
No. Non-Library Modules:	16		
No. Library Modules:	22		
Files:	compress.c		

Program:	**flex**		
Description:	GNU Project lexical scanner generator.		
Version:	04b		
Date:	30/9/87		
Total No. Modules:	124		
No. Non-Library Modules:	96		
No. Library Modules:	28		
Files:	flexccl.c	flexcmp.c	flexdfa.c
	flexecs.c	flexmain.c	flexmisc.c
	flexnfa.c	flexparse.y	flexscan.c
	flexscan.l	flexsum.c	yylex.c

Program:	**prolog**		
Description:	CProlog, a Prolog interpreter written in C for 32-bit machines.		
Version:	1.2		
Total No. Modules:	161		
No. Non-Library Modules:	117		
No. Library Modules:	44		
Files:	arith.c	auxfn.c	compare.c
	dbase.c	main.c	parms.c
	rewrite.c	space.c	sysbits.c
	unify.c		

Program:	**top**		
Description:	Utility that gives continual reports on the status of the system, including a list of the top cpu using processes.		
Version:	2.0		
Total No. Modules:	108		
No. Non-Library Modules:	56		
No. Library Modules:	52		
Files:	bzero.c	commands.c	display.c
	getopt.c	kernel.c	screen.c
	top.c		

Program:	**uemacs-3.10 (MicroEmacs)**		
Description:	MicroEmacs screen based text editor.		
Version:	3.10		
Total No. Modules:	447		
No. Non-Library Modules:	390		
No. Library Modules:	57		
Files:	basic.c	bind.c	buffer.c
	char.c	crypt.c	display.c
	eval.c	exec.c	file.c
	fileio.c	input.c	isearch.c
	line.c	lock.c	main.c
	random.c	region.c	search.c
	tcap.c	unix.c	window.c
	word.c		

Program:	**vn**		
Description:	Visual News reader for visual page oriented display of news aimed at scanning large numbers of articles.		
Total No. Modules:	176		
No. Non-Library Modules:	122		
No. Library Modules:	54		
Files:	digest.c	envir_set.c	getch.c
	hash.c	help.c	newdisp.c
	pagefile.c	printex.c	reader.c
	reg.c	session.c	sig_set.c
	stat.c	std.c	storage.c
	strtok.c	svart.c	term_set.c
	tmpnam.c	tty_set.c	userlist.c
	vn.c		

Program:	**wm**		
Description:	A simple multiple-window monitor for UNIX.		
Date:	28/7/1980		
Total No. Modules:	140		
No. Non-Library Modules:	71		
No. Library Modules:	69		
Files:	cmd.c	curses.c	getch.c
	hacks.c	help.c	misc.c
	save.c	shell.c	vterm.c
	wlist.c	wm.c	

Program:	**xscheme**		
Description:	XSCHEME, object-oriented Scheme interpreter.		
Version:	0.16		
Date:	9/1/89		
Total No. Modules:	502		
No. Non-Library Modules:	467		
No. Library Modules:	35		
Files:	unixstuff.c	xscheme.c	xscom.c
	xsdmem.c	xsftab.c	xsfun1.c
	xsfun2.c	xsimage.c	xsinit.c
	xsint.c	xsio.c	xsmath.c
	xsobj.c	xsprint.c	xsread.c
	xssym.c		

Program:	**yap**		
Description:	Yet Another Pager - text file browser program.		
Version:	6.1		
Date:	5/11/85		
Total No. Modules:	144		
No. Non-Library Modules:	109		
No. Library Modules:	35		
Files:	commands.c	display.c	getcomm.c
	getline.c	help.c	keys.c
	machine.c	main.c	options.c
	output.c	process.c	prompt.c
	term.c		

Appendix D2 - Metric Values For Each Program.

This appendix contains the metric values calculated for each program in the test set of programs. That is, for McCabe' cyclomatic complexity, NPATH, number of statements, structural fanin and fanout, and fanout to library modules metrics.

Program: Bib

Module	McCabe v(G)	NPATH	No. Statements	Fanin	Fanout	Fanout to Lib. Mods.
aabet	6	13	12	1	1	0
aabetlast	3	3	4	1	1	0
addc	2	2	5	1	2	1
alpha_seek	11	360	34	1	6	3
astro	13	47	21	1	1	0
atoi	•	•	•	2	•	•
bibwarning	1	1	3	5	1	1
bldcite	22	22	43	2	7	1
bldname	18	1650	46	1	2	1
breakname	10	192	19	2	0	0
calloc	•	•	•	1	•	•
changefmt	2	4	5	1	1	1
citemark	4	6	7	1	2	1
citesort	1	1	1	1	0	0
cleanup	1	1	5	3	4	4
disambiguate	6	10	10	1	2	2
doargs	55	108	135	1	9	5
doline	11	11	24	1	4	3
dumpref	12	35	33	1	5	3
error	1	1	2	11	2	0
exit	•	•	•	3	•	•
expand	6	8	14	3	5	2
fclose	•	•	•	6	•	•
fetchref	3	4	8	1	2	2
fgets	•	•	•	3	•	•
fhunt	9	28	15	1	2	0
foldline	3	3	3	1	1	1
fopen	•	•	•	5	•	•
fprintf	•	•	•	10	•	•
fputs	•	•	•	5	•	•
fread	•	•	•	1	•	•
free	•	•	•	2	•	•
fscanf	•	•	•	2	•	•
fseek	•	•	•	8	•	•
ftell	•	•	•	5	•	•
fullaabet	6	6	9	1	1	0
fwrite	•	•	•	1	•	•
getfield	21	910	38	3	2	1
getline	4	6	5	1	0	0
getname	12	90	24	5	2	0
getref	12	49	37	1	11	4
getword	8	100	15	0	1	1
getwrd	9	24	9	1	0	0
hunt	10	18	16	1	1	0
incfile	45	1422	109	2	18	10
index	•	•	•	3	•	•
intr	1	1	1	1	1	0
iswordc	4	4	3	1	0	0
load_comm	6	13	14	1	3	3
locate	32	18445700	106	1	17	14
lookup	3	3	5	1	2	2
main	7	448	30	0	15	6
makecites	3	3	7	1	6	3
makekey	9	36	18	1	3	1
malloc	•	•	•	2	•	•
match	7	19	11	0	0	0
mktemp	•	•	•	1	•	•
newbibdir	1	1	5	1	4	2
nextline	2	4	4	1	2	2
nextrecord	5	14	13	0	2	2
pass2	15	49306	26	1	4	0
printline	3	77	3	0	0	0
prtauth	9	48	12	1	5	3
putrefs	29	2862970	70	1	10	5
qsort	•	•	•	2	•	•
rcomp	21	5187	51	1	3	1
rdcite	16	96	36	1	4	1
rdref	1	1	3	5	2	2
rdtext	33	2440	56	2	4	3
recsize	6	18	12	0	2	2
refssearch	4	4	7	1	4	2
rindex	•	•	•	1	•	•
scopy	2	2	3	1	0	0
signal	•	•	•	1	•	•
sprintf	•	•	•	2	•	•
strcat	•	•	•	5	•	•
strcmp	•	•	•	7	•	•

strcpy	•	•	•	12	•	•
strhash	3	4	6	4	0	0
stripkeys	6	8	9	1	1	0
strlen	•	•	•	15	•	•
strncmp	•	•	•	1	•	•
strncpy	•	•	•	1	•	•
strreplace	1	1	3	1	3	3
tfgets	4	4	7	1	1	1
tmpfile	•	•	•	1	•	•
tolower	•	•	•	2	•	•
ungetc	•	•	•	2	•	•
unlink	•	•	•	1	•	•
walloc	2	2	5	2	4	3
wordsearch	4	4	5	2	3	2
wordstuff	2	8	9	2	4	1
wrref	1	1	2	1	2	2

Program: Bison

Module	McCabe v(G)	NPATH	No. Statements	Fanin	Fanout	Fanout to Lib. Mods.
abort	•	•	•	1	•	•
action_row	28	59520	63	1	0	0
add_lookback_edge	5	8	13	1	2	0
allocate_itemsets	5	9	20	1	1	0
allocate_storage	1	1	4	1	2	0
append_states	5	8	10	1	1	0
atoi	•	•	•	1	•	•
augment_automaton	13	50	72	1	3	1
bcopy	•	•	•	1	•	•
berror	1	1	2	3	2	2
build_relations	15	372	45	1	5	1
calloc	•	•	•	1	•	•
closure	14	140	36	1	0	0
compute_FOLLOWS	3	3	5	1	2	1
compute_lookaheads	6	12	16	1	1	1
conflict_log	3	3	9	1	3	0
copys	2	2	6	1	2	1
copy_action	45	494998000	113	2	5	2
copy_definition	26	49053000	67	1	2	1
copy_guard	51	550911000	125	1	8	2
count_rr_conflicts	7	26	20	2	0	0
count_sr_conflicts	12	600	35	2	0	0
default_goto	6	24	15	1	0	0
digraph	5	8	12	2	3	1
done	10	768	20	5	4	3
exit	•	•	•	1	•	•
fatal	2	2	4	17	2	1
fatals	1	1	2	11	2	1
fclose	•	•	•	3	•	•
finalize_closure	1	1	3	1	1	1
finalize_conflicts	1	1	3	0	1	1
flush_shift	5	5	6	1	0	0
fopen	•	•	•	1	•	•
fprintf	•	•	•	39	•	•
free	•	•	•	30	•	•
free_derives	1	1	2	0	1	1
free_itemsets	2	2	4	1	1	1
free_nullable	1	1	1	0	1	1
free_reductions	2	2	4	1	1	1
free_shifts	2	2	4	1	1	1
free_storage	1	1	7	1	1	1
free_symtab	3	3	6	1	1	1
generate_states	3	3	14	1	11	0
gensym	1	1	5	1	2	1
getargs	13	168	35	1	4	3
getenv	•	•	•	2	•	•
getopt	•	•	•	1	•	•
getsym	6	12	21	3	4	1
get_state	10	34	25	1	1	0
get_type	8	12	20	1	6	2
get_type_name	5	8	10	2	1	0
goto_actions	3	57	16	1	5	2
hash	2	2	5	1	0	0
initialize_closure	1	1	4	1	2	0
initialize_conflicts	2	2	7	1	2	0
initialize_F	11	87	33	1	4	1
initialize_LA	13	105	25	1	1	0
initialize_states	1	1	3	1	1	0
init_lex	1	1	1	1	0	0
insert_start_shift	1	1	11	1	1	0
lalr	1	1	12	1	11	0
lex	117	10480	198	6	6	1
log_resolution	1	1	1	1	1	1
main	2	2	15	0	12	0
mallocate	2	2	5	45	3	2
map_goto	4	4	11	2	1	0
matching_state	10	62	16	1	0	0
mktemp	•	•	•	1	•	•
new_itemsets	5	8	15	1	0	0
new_state	3	4	16	1	2	0
openfiles	11	288	45	1	7	5
open_extra_files	2	20	10	1	4	3
output	7	176	22	1	12	3
output_actions	1	1	21	1	11	1
output_base	5	3249	20	1	2	2
output_check	3	57	11	1	2	2
output_defines	1	1	3	1	1	1

output_gram	6	6441	21	1	1	1
output_headers	2	4	3	1	1	1
output_ltype	3	4	10	1	1	1
output_parser	15	542948	33	1	2	2
output_program	2	30	5	1	1	1
output_rule_data	18	1671070000	55	1	2	2
output_stos	3	57	10	1	1	1
output_table	3	57	12	1	2	2
output_token_defines	9	245	11	1	1	1
output_token_translations	5	115	15	1	1	1
output_trailers	2	2	4	1	1	1
packgram	8	25	28	1	1	0
packsymbols	18	2520	43	1	5	2
pack_table	6	24	23	1	4	1
pack_vector	14	434	29	1	2	0
parse_assoc_decl	13	25	35	1	8	3
parse_expect_decl	6	48	10	2	2	2
parse_percent_token	27	2688	52	2	2	2
parse_start_decl	3	4	6	2	2	0
parse_token_decl	13	69	27	2	9	4
parse_type_decl	8	18	19	1	8	3
parse_union_decl	21	920543000	58	2	4	2
perror	•	•	•	1	•	•
print_actions	17	165000	43	1	2	1
print_core	6	5502	19	1	1	1
print_reductions	39	14113000	112	1	1	1
print_state	1	1	3	1	3	1
print_token	1	1	1	1	1	1
reader	2	2	33	1	13	1
readgram	44	3628980	125	1	16	2
read_declarations	18	17	49	1	12	0
read_signed_integer	3	9	9	2	1	1
realloc	•	•	•	1	•	•
record_rule_line	3	3	7	1	3	2
resolve_sr_conflict	19	218	46	1	4	1
rewind	•	•	•	4	•	•
RTC	3	3	12	1	1	0
save_column	6	18	17	1	1	0
save_reductions	6	15	19	1	1	0
save_row	6	18	14	1	1	0
save_shifts	3	4	13	1	1	0
set_accessing_symbol	2	2	3	1	1	0
set_conflicts	14	560	35	1	1	0
set_derives	4	6	20	1	2	1
set_fderives	7	11	19	1	3	1
set_firsts	4	4	12	1	2	0
set_goto_map	10	96	33	1	3	1
set_maxrhs	4	4	9	1	0	0
set_nullable	12	60	42	1	2	1
set_reduction_table	2	2	3	1	1	0
set_shift_table	2	2	3	1	1	0
set_state_table	2	2	3	1	1	0
skip_white_space	15	254	32	7	2	0
sort_actions	9	26	15	1	1	0
sprintf	•	•	•	3	•	•
strcmp	•	•	•	5	•	•
strcpy	•	•	•	5	•	•
stringappend	4	8	13	1	1	0
strlen	•	•	•	5	•	•
tabinit	1	1	3	1	1	0
TC	6	9	21	1	0	0
terse	2	2	2	1	1	0
token_actions	3	57	16	1	5	2
toomany	1	1	2	2	2	1
total_conflicts	9	3969	15	2	1	1
transpose	9	48	23	1	2	1
traverse	10	60	25	2	1	0
tryopen	2	2	6	3	4	3
ungetc	•	•	•	10	•	•
unlex	1	1	2	1	0	0
unlink	•	•	•	1	•	•
verbose	7	20	12	1	4	1
verbose_conflict_log	9	1325	22	1	4	1

Program: Chef

Module	McCabe v(G)	NPATH	No. Statements	Fanin	Fanout	Fanout to Lib. Mods.
abs	•	•	•	1	•	•
accept_char	2	2	4	5	1	0
accept_modifier	2	4	6	4	4	1
accept_pat_or_tag	6	5	13	2	3	0
add_to_tree	9	48	18	1	0	0
alter_case	2	7	5	1	2	2
break_server	1	1	1	1	0	0
brk	2	2	4	3	1	0
buffer_token	2	2	6	1	0	0
calloc	•	•	•	2	•	•
char_number	3	3	4	6	0	0
check_blocks	1	0	0	3	0	0
check_error_flag	1	1	3	1	0	0
check_interrupt	2	2	3	9	1	0
check_lines	25	52800	34	1	1	0
check_screen_full	3	3	7	5	1	0
ci	1	2	1	2	0	0
clear_all_flags	3	3	3	1	2	0
close	•	•	•	2	•	•
closure_length	7	9	18	1	1	0
cmd	5	8	21	6	8	1
co	1	7	1	4	0	0
concat_string	9	108	20	4	0	0
copy_lines	5	24	10	5	2	0
copy_saved_lines	2	2	3	1	2	0
delete_lines	1	1	2	7	1	0
digit_value	7	30	9	1	1	0
do_a	40	3192	108	1	22	1
do_b	3	4	5	1	2	0
do_c	1	1	7	1	2	0
do_cut	14	184	28	1	10	1
do_d	1	2	2	1	1	0
do_e	9	18	16	1	6	0
do_edge	8	23	16	2	7	0
do_f	7	40	24	1	9	4
do_h	14	918	34	1	10	3
do_i	10	11	23	2	6	0
do_j	3	4	16	1	6	0
do_l	25	25	52	1	4	1
do_m	9	82	29	1	7	3
do_mark	11	82	27	1	3	1
do_merge	8	96	27	1	12	1
do_n	1	1	2	1	2	0
do_null	5	5	9	1	1	0
do_p	9	8	23	2	2	0
do_paste	9	126	36	1	12	1
do_q	6	20	12	1	5	2
do_query	16	18	44	2	2	1
do_r	5	9	8	1	3	0
do_return	4	24	25	1	14	0
do_scr	14	30	44	17	4	1
do_sys_call	3	3	6	1	4	4
do_t	2	2	6	1	3	0
do_u	3	4	10	1	4	2
do_v	5	64	14	1	1	0
do_w	2	2	6	1	2	0
do_x	7	108	20	1	5	1
do_xa	11	30	43	1	9	2
do_xc	1	1	1	1	1	0
do_xf	5	16	16	1	6	3
dsply_lines	3	8	9	10	4	0
dsply_range	3	8	9	5	1	0
empty_work_space	11	35	21	2	5	0
ensure_ws_consistent	4	4	8	5	1	0
exit	•	•	•	1	•	•
expand	1	1	2	7	1	0
extend_str	5	12	8	2	0	0
fclose	•	•	•	6	•	•
fetch_grab	1	1	1	9	1	0
fetch_line	1	2	2	18	2	0
fetch_record	4	4	17	3	1	0
fetch_saved_line	2	2	4	2	1	0
fgetc	•	•	•	3	•	•
find_block	5	12	16	2	1	0
find_bracket	23	9072	53	1	8	1
first_pos	2	4	4	1	0	0
flagged_lines	7	14	18	1	5	0

flag_the_line	1	1	1	1	2	0
flush_buffer	7	6	16	2	2	0
fopen	•	•	•	4	•	•
fprintf	•	•	•	1	•	•
fseek	•	•	•	1	•	•
getenv	•	•	•	2	•	•
gets_file	8	20	17	1	1	1
gets_scroll	9	48	20	1	3	1
getword	10	32	18	1	1	0
get_all	3	3	6	1	2	0
get_ch	3	3	7	22	0	0
get_class	17	1600	50	1	3	0
get_clip_head	8	480	24	1	4	3
get_finder	10	20	22	1	4	0
get_first_key	4	6	7	1	0	0
get_fm_opnd	6	10	19	1	5	0
get_hlp_directory	9	102	29	1	5	4
get_int	2	2	4	5	1	0
get_j_opnd	6	18	11	1	2	0
get_left_lines	1	1	3	1	1	0
get_location	12	240	35	2	3	0
get_lspec	5	5	12	1	6	1
get_l_opnd	10	9	16	1	4	0
get_name	4	4	8	1	2	0
get_new	6	7	18	1	3	0
get_pat	16	291	46	3	5	0
get_range	3	4	8	1	4	0
get_right_lines	1	1	3	1	1	0
get_scr	17	17	33	1	3	0
get_screen_input	34	113	111	1	7	0
get_term	6	16	13	1	7	0
get_x_opnd	7	12	17	1	5	0
got_text	7	20	14	6	3	0
go_aft	6	6	18	1	4	0
go_fore	4	4	13	1	3	0
grab_ad	2	2	7	2	1	0
guard_file	1	1	3	2	2	1
init_buffers	5	5	21	2	1	1
init_pattern	1	1	10	3	0	0
init_work_space	6	16	9	3	3	1
input_ch	13	50	36	1	2	1
insert_blank_line	3	3	8	1	5	0
insert_lines	11	58	28	2	5	0
internal_code	2	2	8	4	2	0
interpret	31	104	86	1	29	0
in_class	5	7	9	2	0	0
ioctl	•	•	•	1	•	•
is_line	2	2	4	1	1	0
key_pos	6	13	13	1	0	0
loaded_ptrs	2	2	3	2	1	0
loaded_text	2	2	4	2	1	0
longjmp	•	•	•	3	•	•
lseek	•	•	•	2	•	•
lsubstitute	6	26	12	1	5	0
main	11	32	29	0	14	4
make_key	4	8	7	1	1	1
make_space	2	2	7	4	2	0
malloc	•	•	•	1	•	•
mktemp	•	•	•	1	•	•
move_windows	5	16	21	3	2	0
next_sig_char	2	2	2	7	1	0
next_tab	3	6	7	1	0	0
next_word	3	3	5	1	0	0
omatch	8	12	14	1	2	1
open	•	•	•	1	•	•
open_work_file	4	16	9	2	3	3
operand	58	31920	129	1	20	2
operator	11	648	17	1	5	1
out_lines	10	270	22	1	8	3
postlude	3	4	5	1	3	3
post_trail	26	34944	45	1	3	2
pre_trail	5	8	8	1	2	0
printable	1	1	4	4	0	0
printf	•	•	•	19	•	•
print_count	3	3	5	2	1	1
process_line	24	64512	45	1	6	1
put	7	144	17	1	3	1
puts_screen	5	9	5	5	1	0
putword	9	100	24	1	2	0
put_legend	5	4	17	1	1	0

put_msg	2	2	3	3	1	1
qmatch	4	5	10	1	0	0
read	•	•	•	1	•	•
readn	8	48	11	1	1	1
read_file	3	4	10	2	6	3
recognize	5	6	10	1	0	0
recover	6	32	13	1	4	2
recover_eof	1	1	2	1	1	0
remove_lines	5	12	12	2	2	0
replace	6	9	19	1	4	3
restore_block	1	1	2	1	2	2
rmatch	10	385	21	2	3	0
save_block	1	1	2	1	2	2
save_clip	2	2	4	1	2	0
save_cmd	9	8	12	1	0	0
scan_file	12	150	26	1	4	0
scan_line	15	366	28	4	2	0
scr_token	4	4	7	1	1	0
setjmp	•	•	•	1	•	•
setup_attention	3	4	5	1	2	1
setup_editor	1	1	29	1	3	1
setup_environment	1	1	1	1	1	0
setup_screen	50	0	129	1	8	7
setup_workspace	2	2	13	1	4	0
set_borders	1	1	2	3	0	0
set_cursor	1	1	4	9	4	3
set_tabs	2	3	3	2	0	0
set_terminal	2	2	10	1	1	1
set_window	1	6	4	11	3	0
set_wrap	1	1	1	5	0	0
show_line	12	5376	27	2	3	1
signal	•	•	•	1	•	•
spread	6	18	14	1	1	1
sprintf	•	•	•	4	•	•
sscanf	•	•	•	1	•	•
stack_work_space	1	1	3	1	2	1
store_grab	1	1	1	9	1	0
store_line	2	2	3	13	2	0
store_new_record	5	8	22	3	3	1
strcat	•	•	•	4	•	•
strcmp	•	•	•	2	•	•
strcpy	•	•	•	23	•	•
strlen	•	•	•	11	•	•
strncpy	•	•	•	1	•	•
swapped_workspace	7	48	42	1	8	3
swap_block	3	5	5	1	3	0
system	•	•	•	1	•	•
tabulate	8	14	18	1	0	0
terminal_action	53	3118	98	2	8	2
tgetent	•	•	•	1	•	•
tgetnum	•	•	•	2	•	•
tgetstr	•	•	•	1	•	•
tgoto	•	•	•	1	•	•
tolower	•	•	•	1	•	•
toupper	•	•	•	6	•	•
tputs	•	•	•	2	•	•
uc_char_number	2	2	3	0	2	1
unget_ch	1	1	3	2	0	0
unlink	•	•	•	2	•	•
unstack_work_space	2	2	8	2	4	2
warn	6	5	12	36	3	1
was_flagged	2	2	5	1	2	0
window	3	3	9	2	5	0
window_parameters	9	200	33	2	5	1
window_shift	3	6	7	3	0	0
wp_cmd	6	6	6	1	0	0
wrap_column	3	4	6	5	0	0
wrap_pos	5	8	9	1	0	0
write	•	•	•	1	•	•
write_out	5	20	16	3	7	3

Program: Compress

Module	McCabe v(G)	NPATH	No. Statements	Fanin	Fanout	Fanout to Lib. Mods.
atoi	•	•	•	1	•	•
chmod	•	•	•	1	•	•
chown	•	•	•	1	•	•
cl_block	4	6	14	1	2	0
cl_hash	3	4	21	2	0	0
compress	17	6430320	60	1	6	1
copystat	13	60	33	1	8	8
decompress	14	86240	38	1	3	1
exit	•	•	•	4	•	•
fclose	•	•	•	1	•	•
fflush	•	•	•	3	•	•
foreground	3	3	5	1	1	1
fprintf	•	•	•	7	•	•
fread	•	•	•	1	•	•
freopen	•	•	•	1	•	•
fwrite	•	•	•	1	•	•
getcode	9	30	28	1	1	1
isatty	•	•	•	1	•	•
main	62	262130000	161	0	23	14
malloc	•	•	•	1	•	•
onintr	2	2	3	1	2	2
oops	3	4	5	1	3	3
output	14	400	41	2	3	2
perror	•	•	•	3	•	•
prratio	3	16	7	1	1	1
read	•	•	•	1	•	•
rindex	3	3	4	1	0	0
signal	•	•	•	1	•	•
stat	•	•	•	2	•	•
strcat	•	•	•	1	•	•
strcmp	•	•	•	1	•	•
strcpy	•	•	•	1	•	•
strlen	•	•	•	1	•	•
unlink	•	•	•	4	•	•
Usage	1	1	1	1	1	1
utime	•	•	•	1	•	•
version	1	1	3	1	1	1
writeerr	1	1	3	3	3	3

Program **Flex**

Module	McCabe v(G)	NPATH	No. Statements	Fanin	Fanout	Fanout to Lib. Mods.
abs	•	•	•	1	•	•
action_out	4	4	4	1	2	2
addsym	5	12	16	3	4	2
add_accept	8	18	17	1	4	1
allocate_array	2	2	3	4	2	1
bldtbl	12	81	32	1	5	0
bubble	4	4	6	1	0	0
bzero	•	•	•	2	•	•
ccl2ecl	4	4	10	1	0	0
ccladd	4	6	12	1	1	0
cclinit	3	4	12	1	1	0
cclinstal	1	1	1	0	2	0
cllookup	1	1	1	0	1	0
cclnegate	1	1	1	1	0	0
clower	1	2	1	3	1	1
cmptmps	9	28	19	1	3	0
copysingl	2	2	4	2	3	0
copy_string	4	8	8	4	2	1
cre8ecs	4	4	7	2	0	0
cshell	5	5	9	1	0	0
ctime	•	•	•	1	•	•
dataend	2	2	4	2	2	1
dataflush	2	56	5	3	0	0
dumpnfa	3	3	12	1	1	1
dupmachine	5	5	13	2	2	0
epsclosure	24	12978	72	1	3	0
exit	•	•	•	6	•	•
expand_nxt_chk	1	1	5	4	2	1
fclose	•	•	•	2	•	•
fflush	•	•	•	0	•	•
fgets	•	•	•	2	•	•
findsym	3	3	5	1	2	1
find_table_space	14	675	26	1	1	0
flexend	7	34	38	3	5	4
flexerror	1	1	2	3	2	1
flexfatal	1	1	2	10	2	1
flexinit	49	10990100	127	1	10	4
flexscan	151	35955	597	2	6	2
fopen	•	•	•	2	•	•
fprintf	•	•	•	10	•	•
fputs	•	•	•	13	•	•
free	•	•	•	2	•	•
freopen	•	•	•	1	•	•
fwrite	•	•	•	0	•	•
genctbl	9	48	31	1	2	1
gentabs	46	-135138000	109	1	8	4
getdef	2	2	3	3	1	0
gettime	1	1	4	2	3	2
hashfunct	2	2	5	2	0	0
increase_max_dfas	3	3	14	3	1	0
inittbl	3	3	10	1	1	1
input	9	15	27	1	6	2
lerrif	1	1	2	3	2	1
lerrsf	1	1	2	2	2	1
line_directive_out	3	3	2	3	1	1
link_machines	3	6	9	7	1	0
main	2	2	6	0	5	0
make_tables	5	18	16	1	9	3
malloc	•	•	•	6	•	•
mk1tbl	6	18	13	2	1	0
mk2data	3	64	8	1	3	2
mkbranch	3	3	8	2	2	0
mkclos	1	1	1	2	2	0
mkdata	3	64	8	1	3	2
mkdeftbl	3	4	11	1	1	0
mkeccl	12	91	26	3	0	0
mkechar	3	4	6	2	0	0
mkentry	31	612000	49	3	2	0
mkopt	3	3	7	3	3	0
mkor	9	9	19	1	3	0
mkposcl	3	3	6	2	3	0
mkprot	5	12	14	2	0	0
mkrep	3	3	10	1	6	0
mkstate	6	9	24	10	3	0
mktemp	•	•	•	1	•	•
mktemplate	5	12	17	1	5	0
mkxtion	4	4	6	6	1	0

mv2front	4	5	10	1	0	0
myctoi	1	1	2	0	1	1
myesc	10	10	26	0	2	0
ndinstal	2	2	2	0	3	0
ndlookup	1	1	1	0	1	0
ntod	41	48798300	121	1	20	3
otoi	1	1	2	1	1	1
place_state	4	6	10	1	1	0
printf	•	•	•	8	•	•
puts	•	•	•	2	•	•
read	•	•	•	1	•	•
readin	10	512	25	1	8	2
realloc	•	•	•	2	•	•
reallocate_array	2	2	3	8	2	1
scinstal	4	8	14	1	7	2
sclookup	1	1	1	1	1	0
set_up_initial_allocations	1	1	34	1	1	0
skelout	4	4	4	2	2	2
snstods	18	1920	45	1	4	1
sprintf	•	•	•	2	•	•
sscanf	•	•	•	2	•	•
stack1	2	2	7	1	1	0
strcmp	•	•	•	3	•	•
strcpy	•	•	•	1	•	•
strlen	•	•	•	0	•	•
symfollowset	15	14	32	1	1	0
sympartition	12	58	29	1	3	0
synerr	1	1	2	4	1	1
tbldiff	3	3	7	2	0	0
time	•	•	•	1	•	•
tolower	•	•	•	1	•	•
transition_struct_out	3	3	7	1	1	1
unlink	•	•	•	1	•	•
write	•	•	•	0	•	•
yyerror	1	0	0	1	0	0
yylex	93	2616	140	1	4	2
yyparse	112	17998200	303	1	24	3
yyrestart	1	1	2	0	2	0
yyunput	7	57	21	0	2	2
yy_create_buffer	5	81	15	1	4	3
yy_delete_buffer	2	2	4	0	1	1
yy_get_next_buffer	12	4914	35	1	3	3
yy_get_previous_state	6	14	13	1	0	0
yy_init_buffer	1	1	7	4	0	0
yy_load_buffer_state	1	1	4	4	0	0
yy_switch_to_buffer	3	4	9	0	1	0
yy_try_NUL_trans	4	12	10	0	0	0

Program: MicroEmacs

Module	McCabe v(G)	NPATH	No. Statements	Fanin	Fanout	Fanout to Lib. Mods.
absv	1	2	1	2	0	0
addline	4	8	12	1	2	1
adjustmode	13	480	31	4	8	6
amatch	17	306	43	2	3	0
anycb	4	4	6	1	0	0
apro	2	2	4	1	2	0
asc_int	8	48	14	6	0	0
atoi	•	•	•	1	•	•
backchar	5	8	11	18	1	0
backdel	6	24	11	5	5	0
backhunt	11	160	15	2	5	1
backline	6	24	14	2	3	0
backpage	10	144	25	3	3	0
backsearch	8	22	10	2	5	1
backword	8	40	12	3	4	0
bclear	6	16	12	8	2	0
bfind	11	123	40	19	5	4
binary	4	5	11	3	1	1
bindtokey	13	1512	42	1	10	1
biteq	2	4	3	1	0	0
bktoshell	1	1	3	1	3	3
boundry	2	2	4	3	0	0
buildlist	28	292608	84	2	12	5
bytecopy	3	3	5	8	0	0
capword	12	256	25	1	8	0
cbuf	4	6	9	40	3	1
cbuf1	1	1	1	0	1	0
cbuf10	1	1	1	0	1	0
cbuf11	1	1	1	1	1	0
cbuf12	1	1	1	1	1	0
cbuf13	1	1	1	1	1	0
cbuf14	1	1	1	1	1	0
cbuf15	1	1	1	1	1	0
cbuf16	1	1	1	1	1	0
cbuf17	1	1	1	1	1	0
cbuf18	1	1	1	1	1	0
cbuf19	1	1	1	1	1	0
cbuf2	1	1	1	1	1	0
cbuf20	1	1	1	1	1	0
cbuf21	1	1	1	1	1	0
cbuf22	1	1	1	1	1	0
cbuf23	1	1	1	1	1	0
cbuf24	1	1	1	1	1	0
cbuf25	1	1	1	1	1	0
cbuf26	1	1	1	1	1	0
cbuf27	1	1	1	1	1	0
cbuf28	1	1	1	1	1	0
cbuf29	1	1	1	1	1	0
cbuf3	1	1	1	1	1	0
cbuf30	1	1	1	1	1	0
cbuf31	1	1	1	1	1	0
cbuf32	1	1	1	1	1	0
cbuf33	1	1	1	1	1	0
cbuf34	1	1	1	1	1	0
cbuf35	1	1	1	1	1	0
cbuf36	1	1	1	1	1	0
cbuf37	1	1	1	1	1	0
cbuf38	1	1	1	1	1	0
cbuf39	1	1	1	1	1	0
cbuf4	1	1	1	1	1	0
cbuf40	1	1	1	1	1	0
cbuf5	1	1	1	1	1	0
cbuf6	1	1	1	1	1	0
cbuf7	1	1	1	1	1	0
cbuf8	1	1	1	1	1	0
cbuf9	1	1	1	1	1	0
cclmake	13	168	39	1	4	2
cex	1	1	5	2	0	0
chcase	3	4	5	3	2	0
checknext	5	7	16	1	2	0
chmod	•	•	•	1	•	•
chown	•	•	•	1	•	•
cinsert	13	480	24	1	4	0
clearbits	3	3	4	1	1	1
closedir	•	•	•	1	•	•
clrmes	1	1	2	0	1	1
cmdstr	8	128	24	7	0	0

complete	65	1065	153	4	14	7
comp_buffer	11	79	28	1	0	0
comp_command	11	79	30	1	0	0
comp_file	13	324	29	1	5	3
copyregion	7	20	18	1	4	1
crypt	9	30	20	5	1	0
ctime	•	•	•	1	•	•
ctlxe	3	4	9	0	1	1
ctlxlp	2	2	8	0	1	1
ctlxrp	3	4	7	0	1	1
ctoec	3	3	3	4	0	0
ctrlg	1	1	4	3	1	1
dcline	41	4158	101	1	26	4
deblank	7	36	14	0	2	0
debug	12	20	83	1	14	4
delbword	11	320	23	1	6	0
delfword	21	4512	40	1	5	0
delgmode	1	1	1	1	1	0
delins	8	8	11	1	3	1
delline	1	1	16	0	6	0
delmode	1	1	1	0	1	0
delwind	14	432	45	0	3	3
desbind	1	1	1	0	1	0
desfunc	9	108	40	0	8	4
deskey	2	2	7	0	6	1
desvars	12	540	52	0	11	5
detab	6	32	18	0	7	0
dispvar	4	6	16	0	11	5
dobuf	81	-1173180000	203	6	14	6
docmd	9	84	42	3	11	3
dofile	5	16	17	2	6	0
echochar	7	6	30	1	2	2
ectoc	3	4	5	4	0	0
edinit	5	8	34	1	5	1
editloop	30	20400	64	1	9	3
endword	7	20	10	0	3	0
enlargewind	11	288	28	3	2	1
entab	14	416	43	0	9	0
envval	1	1	1	1	0	0
eq	4	5	6	5	2	0
ernd	1	1	2	2	1	0
execbuf	5	12	9	0	4	1
execcmd	2	2	4	2	2	0
execfile	6	18	11	1	4	1
execkey	5	8	8	6	1	0
execprg	5	12	16	0	6	3
execproc	5	12	11	1	5	2
execute	25	15778	33	1	11	2
exit	•	•	•	1	•	•
expandp	7	11	21	3	0	0
extcode	1	1	1	0	0	0
fbound	12	28	39	1	0	0
fclose	•	•	•	3	•	•
fexist	2	2	5	2	2	2
ffclose	3	4	7	4	3	3
ffgetline	14	1260	29	2	6	4
fflush	•	•	•	1	•	•
ffputline	5	224	12	1	2	1
ffropen	2	2	4	3	1	1
ffwopen	2	2	4	1	2	2
fgets	•	•	•	1	•	•
filefind	3	4	5	0	3	0
filename	4	8	10	1	4	2
fileread	3	4	5	0	3	0
filesave	9	144	19	2	5	2
filewrite	4	8	9	0	5	2
fillpara	11	448	45	1	11	3
filter	7	48	33	0	10	5
findvar	12	23	32	3	5	2
fisearch	2	2	10	0	4	3
fixnull	2	2	3	8	0	0
flook	21	7488	43	4	6	4
fmatch	12	234	29	1	3	1
fncmatch	2	2	4	4	2	0
fopen	•	•	•	4	•	•
forwchar	5	8	11	19	1	0
forwdel	5	12	9	2	4	0
forwhunt	11	160	15	2	5	1
forwline	6	24	14	8	3	0
forwpage	10	144	25	2	3	0

forwsearch	8	22	10	2	5	1
forwword	7	20	10	2	3	0
fputc	•	•	•	1	•	•
free	•	•	•	20	•	•
freewhile	2	2	3	2	2	1
funval	1	1	1	1	0	0
get1key	13	270	33	1	2	1
getbind	3	3	6	6	0	0
getcbuf	2	2	4	2	2	0
getccol	8	17	11	12	0	0
getckey	3	4	7	4	4	0
getcline	3	3	8	1	0	0
getcmd	4	4	12	3	5	0
getctext	3	4	10	1	0	0
getdefb	5	6	9	3	0	0
getenv	•	•	•	6	•	•
getfence	18	2772	60	1	3	0
getffile	10	120	23	1	5	4
getfile	15	378	43	4	8	3
getfname	7	24	17	3	0	0
getgoal	6	9	15	6	0	0
getkey	5	9	11	7	1	0
getkill	3	3	7	1	1	0
getname	2	2	4	2	2	0
getnfile	5	12	11	2	4	4
getpid	•	•	•	1	•	•
getpwnam	•	•	•	1	•	•
getreg	5	12	16	1	1	0
getregion	10	66	34	8	1	1
getstring	25	202	54	3	7	3
gettyp	13	54	24	2	0	0
getval	18	43	50	7	11	1
getwpos	2	2	6	2	0	0
get_char	4	6	12	1	3	2
gotobob	1	1	4	1	0	0
gotobol	1	1	2	2	0	0
gotobop	11	92	16	4	4	0
gotobos	1	1	4	0	0	0
gotoeob	1	1	4	1	0	0
gotoeol	1	1	2	1	0	0
gotoeop	12	182	19	4	4	0
gotoeos	3	3	8	0	0	0
gotoline	4	6	10	2	4	1
gotomark	3	4	10	1	1	1
gtenv	64	126	131	2	16	0
gtfilename	2	2	4	4	1	0
gtfun	51	1290	112	1	32	7
gtpattern	2	2	5	1	2	1
gtty	•	•	•	1	•	•
gtusr	5	7	9	2	1	1
help	6	18	16	1	7	2
ifile	13	2560	61	1	10	4
indent	13	172	16	0	3	0
initchars	4	8	7	1	0	0
insbrace	21	10200	50	1	6	0
insfile	4	8	7	1	4	0
inspound	6	16	9	1	3	0
insspace	2	2	3	2	2	0
int_asc	4	8	13	6	0	0
inword	5	12	8	10	1	0
in_check	2	2	3	1	0	0
in_get	1	1	3	1	0	0
in_init	1	1	1	1	0	0
in_put	1	1	2	1	0	0
ioctl	•	•	•	4	•	•
isearch	26	1267	71	2	10	1
isent	9	22	9	1	1	1
isletter	1	1	1	3	2	0
islower	1	1	1	8	0	0
istring	6	24	10	1	2	0
isupper	1	1	1	6	0	0
kdelete	3	3	8	7	1	1
kill	•	•	•	1	•	•
killbuffer	2	4	5	1	3	0
killpara	3	3	10	1	4	0
killregion	4	8	10	2	4	0
killtext	9	28	23	1	4	1
kinsert	5	9	12	2	1	1
lalloc	2	2	6	8	2	2
lchange	5	12	10	13	0	0

lckhello	1	0	0	0	0	0
ldelete	21	6722	42	14	4	0
ldelnewline	22	47952	62	1	4	1
lfree	11	104	26	3	1	1
link	•	•	•	1	•	•
linsert	20	11880	61	12	5	2
linstr	5	7	8	8	3	1
listbuffers	9	56	27	0	2	0
lnewline	12	400	34	12	3	0
long_asc	3	4	7	1	0	0
lover	5	7	8	1	3	1
lowerc	2	2	3	3	1	0
lowercase	2	2	2	1	1	0
lowerregion	5	12	16	0	4	0
lowerword	9	64	17	1	6	0
lowrite	4	4	3	1	2	0
ltos	2	2	3	2	0	0
macarg	1	1	5	5	2	0
macrotokey	7	36	35	0	8	3
main	2	2	12	0	98	3
makelist	23	290340	65	1	4	1
makelit	5	6	10	4	0	0
makename	7	24	11	4	0	0
malloc	•	•	•	12	•	•
match_pat	5	7	13	1	2	1
mcclear	4	4	7	3	1	1
mceq	11	10	22	1	5	1
mcscanner	4	4	19	5	4	0
mcstr	17	78	60	3	2	0
meexit	1	1	3	5	0	0
meta	1	1	5	3	0	0
mgetent	12	328	30	1	7	6
mgetstr	33	4536	79	1	2	2
mklower	2	2	4	1	1	0
mkupper	2	2	4	1	1	0
mlerase	•	•	•	7	•	•
mlforce	•	•	•	6	•	•
mlout	•	•	•	5	•	•
mlputs	•	•	•	2	•	•
mlreply	1	1	1	21	2	0
mltreply	1	1	1	2	1	0
mlwrite	•	•	•	85	•	•
mlyesno	8	25	12	3	5	3
mod95	5	16	9	1	0	0
movecursor	•	•	•	5	•	•
mvdnwind	1	1	1	0	1	0
mvupwind	11	90	21	1	0	0
namebuffer	5	8	13	1	4	3
namedcmd	5	10	17	2	8	3
namval	1	1	1	1	0	0
narrow	10	256	41	0	3	1
newline	13	480	16	0	7	0
newsize	14	108	42	1	2	2
newwidth	5	12	14	1	1	1
nextarg	3	4	7	3	4	1
nextbuffer	6	20	12	1	2	0
nextch	4	4	16	4	0	0
nextdown	1	1	3	1	3	0
nextup	1	1	3	1	3	0
nextwind	8	18	20	4	2	2
nullproc	1	0	0	1	0	0
onlywind	9	48	32	1	1	1
opendir	•	•	•	1	•	•
openline	7	48	12	0	3	0
ostring	3	3	3	4	1	1
outstring	3	3	3	1	1	1
ovstring	6	24	10	1	2	0
pad	2	2	3	1	2	2
pipecmd	11	448	36	0	10	3
prevwind	4	8	12	3	2	1
promptpattern	2	2	7	1	5	4
putctext	3	4	8	1	4	0
putnpad	1	1	1	0	2	1
putpad	1	1	1	6	1	1
puts	•	•	•	1	•	•
qreplace	1	1	1	1	1	0
quickexit	5	5	13	1	3	1
quit	5	5	6	1	4	1
quote	4	8	8	0	3	0
rdonly	1	1	3	30	1	1

read	•	•	•	1	•	•
readdir	•	•	•	1	•	•
readin	16	7680	59	5	12	4
readpattern	6	9	13	3	8	1
reeat	2	2	5	1	1	0
reform	9	10	20	1	0	0
refresh	2	2	6	0	0	0
reglines	3	4	12	3	1	0
remmark	2	2	7	0	1	1
rename	1	1	2	1	2	2
replaces	35	184704	98	2	14	5
reposition	2	2	5	1	0	0
resetkey	4	4	12	3	3	1
resize	3	4	6	2	1	0
resterr	1	1	3	11	1	1
restwnd	3	3	10	1	2	2
risearch	2	2	11	0	5	3
rmcclear	3	3	6	1	1	1
rmcstr	13	40	44	1	3	2
rtfrmshell	1	1	3	1	0	0
rvstrcpy	2	2	4	1	1	1
savematch	4	6	9	2	3	2
savewnd	1	1	2	1	0	0
scanmore	3	4	8	1	2	0
scanner	5	9	25	6	4	0
select	•	•	•	1	•	•
setbit	2	2	2	1	0	0
setbuffer	•	•	•	1	•	•
setccol	6	9	13	1	0	0
setekey	2	2	11	2	5	3
setfillcol	1	1	3	0	1	1
setgmode	1	1	1	1	1	0
setjtable	7	40	20	4	4	1
setkey	3	3	5	1	1	0
setlower	1	1	1	1	0	0
setmark	2	2	7	2	1	1
setmod	1	1	1	1	1	0
setupper	1	1	1	1	0	0
setvar	9	72	28	0	13	6
showcpos	6	16	27	0	2	1
shrinkwind	11	288	28	1	2	1
signal	•	•	•	1	•	•
sindex	5	7	13	1	1	0
sleep	•	•	•	1	•	•
spal	1	0	0	1	0	0
spawn	4	8	17	0	6	3
spawncli	4	6	12	0	5	4
splitwind	13	416	53	6	2	2
sprintf	•	•	•	1	•	•
sreplace	1	1	1	1	1	0
startup	3	4	6	1	2	0
stat	•	•	•	2	•	•
stock	19	17496	32	3	1	0
stol	3	4	5	3	1	0
storemac	5	12	16	2	4	2
storeproc	4	8	13	1	6	2
strcat	•	•	•	23	•	•
strcmp	•	•	•	11	•	•
strcpy	•	•	•	40	•	•
strinc	5	7	12	1	0	0
strlen	•	•	•	26	•	•
strncmp	•	•	•	3	•	•
strncpy	•	•	•	2	•	•
stty	•	•	•	2	•	•
svar	71	71	217	2	26	5
swapmark	3	4	14	0	1	1
swbuffer	10	42	37	5	2	0
system	•	•	•	5	•	•
tab	7	36	15	1	3	0
tcapbeep	1	1	1	0	1	0
tcapclose	2	2	3	0	2	0
tcapcres	1	1	1	0	0	0
tcapeeol	1	1	1	0	1	0
tcapeeop	1	1	1	0	1	0
tcapgetc	5	16	16	1	5	0
tcapkclose	1	0	0	0	0	0
tcapkopen	1	1	1	0	1	1
tcapmove	1	1	1	0	2	1
tcapopen	21	163840	57	0	15	8
tcaprev	4	4	5	0	1	0

tgetc	6	15	17	7	1	1
tgetent	•	•	•	1	•	•
tgetnum	•	•	•	1	•	•
tgetstr	•	•	•	1	•	•
tgoto	•	•	•	1	•	•
time	•	•	•	1	•	•
timeset	1	1	4	1	3	3
toggleovermode	2	2	4	1	1	1
token	21	294	45	6	0	0
tputs	•	•	•	2	•	•
transbind	2	2	4	1	3	0
trim	7	40	19	0	4	0
trimstr	4	4	5	1	1	1
ttclose	1	1	2	1	2	2
ttflush	1	1	1	0	1	1
ttgetc	1	1	2	1	1	1
ttopen	1	1	12	1	6	5
ttputc	1	1	1	2	1	1
twiddle	5	12	14	0	2	0
typahead	1	2	1	1	1	1
unarg	1	0	0	1	0	0
unbindchar	5	12	20	2	0	0
unbindkey	2	2	8	1	5	1
uneat	1	1	4	1	0	0
unlink	•	•	•	4	•	•
unmark	1	1	3	1	1	1
unqname	6	9	8	2	1	0
update	•	•	•	11	•	•
upmode	•	•	•	15	•	•
upperc	2	2	3	5	1	0
uppercase	2	2	2	3	1	0
upperregion	5	12	16	0	4	0
upperword	9	64	17	1	6	0
upscreen	•	•	•	1	•	•
upwind	•	•	•	1	•	•
usebuffer	3	8	7	0	3	0
varclean	3	3	3	0	1	1
varinit	2	2	2	1	0	0
viewfile	4	8	9	0	4	1
vtinit	•	•	•	1	•	•
vttidy	•	•	•	2	•	•
widen	8	54	30	0	1	1
wordcount	7	124	27	1	2	1
wpopup	5	9	6	1	1	0
wrapword	11	288	21	1	5	0
writemsg	2	2	5	1	3	1
writeout	16	2268	49	3	17	7
xlat	4	4	14	1	0	0
yank	11	96	21	0	3	0
zotbuf	5	16	17	4	3	2

Program: Prolog

Module	McCabe v(G)	NPATH	No. Statements	Fanin	Fanout	Fanout to Lib. Mods.
abort	•	•	•	1	•	•
access	•	•	•	1	•	•
acos	•	•	•	1	•	•
apply	2	2	11	3	1	0
arg	5	7	10	4	1	0
argv	3	3	8	9	1	0
ArithError	1	1	2	7	1	0
asin	•	•	•	1	•	•
atan	•	•	•	1	•	•
AtomToFile	3	3	3	4	1	0
bread	2	2	10	1	3	0
CallSystem	2	2	3	1	2	1
CatchSignals	1	1	2	1	2	1
CClose	2	2	6	6	1	1
ClearMem	2	2	2	2	0	0
CloseFiles	4	4	3	1	1	0
comp	9	1152	18	2	4	1
compare	1	1	4	1	2	0
ConsFloat	1	1	3	2	0	0
COpen	5	12	9	2	3	1
CopyMem	4	4	7	0	0	0
cos	•	•	•	1	•	•
CreateStacks	5	12	15	1	3	3
CSee	1	1	1	2	1	0
CTell	1	1	1	1	1	0
deref	2	2	3	2	0	0
digits	3	4	7	1	0	0
erase	7	128	23	1	0	0
erased	2	2	5	1	0	0
eval	49	427140	131	4	13	2
Event	5	12	12	8	6	4
execl	•	•	•	1	•	•
Exists	1	1	1	1	1	1
exit	•	•	•	4	•	•
exp	•	•	•	1	•	•
fclose	•	•	•	3	•	•
fentry	6	16	22	2	2	0
ffail	1	1	1	1	12	11
fflush	•	•	•	2	•	•
floor	•	•	•	1	•	•
fopen	•	•	•	3	•	•
ForceInt	3	3	6	1	1	0
fork	•	•	•	1	•	•
fprintf	•	•	•	6	•	•
fputs	•	•	•	2	•	•
fread	•	•	•	1	•	•
fwrite	•	•	•	1	•	•
GarbageCollect	4	16	11	1	3	0
Get	5	24	11	2	3	0
getenv	•	•	•	2	•	•
GetExact	2	2	5	1	0	0
GetMixed	4	4	13	1	1	0
getsp	10	30	24	5	8	1
GetTop	2	2	6	1	0	0
globalize	8	30	32	2	1	0
gunify	115	-1265460000	267	3	2	0
heapify	12	600	33	3	4	0
heapifybody	5	12	13	2	4	0
HeapTop	1	1	1	1	0	0
HeapUsed	1	1	1	1	0	0
InitHeap	1	1	3	1	1	0
InitIO	2	2	14	1	0	0
instance	8	44	39	1	2	0
Interrupt	8	7	21	1	5	2
intsign	1	64	3	1	1	0
intval	3	4	7	1	4	0
IODie	1	1	2	3	1	0
IOError	1	1	1	3	2	0
isop	6	32	18	2	0	0
legalatom	10	144	16	1	1	1
link	•	•	•	1	•	•
list_to_string	11	291	19	2	2	0
log	•	•	•	1	•	•
log10	•	•	•	1	•	•
longjmp	•	•	•	1	•	•
lookup	6	24	27	2	5	3
lookupvar	5	12	20	1	3	3

LookVar	4	6	9	1	0	0
main	435	0	1316	0	69	8
makelist	4	6	21	3	0	0
MergeGarbage	6	12	14	1	0	0
Narrow	4	4	7	4	0	0
nextch	3	4	7	2	0	0
NoSpace	1	3	5	7	2	1
NotInt	1	1	2	1	2	1
NotNumber	1	1	1	1	1	0
NotOp	1	1	2	1	2	1
NumberString	15	3888	34	2	4	1
numcompare	19	210	39	1	2	0
numeval	3	4	7	1	4	0
op	21	55296	43	1	4	1
patom	5	9	10	1	3	0
PClose	6	10	10	1	1	0
perror	•	•	•	2	•	•
pow	•	•	•	1	•	•
pread	25	103680	62	2	6	1
printf	•	•	•	1	•	•
Prompt	1	1	2	1	2	2
PromptIfUser	2	2	3	3	2	1
Put	2	14	3	4	0	0
puts	•	•	•	3	•	•
PutString	2	2	2	4	1	0
pwrite	34	19535000	88	2	8	1
readargs	5	9	11	1	5	0
readlist	8	8	18	1	4	1
record	38	238879000	108	1	7	0
recorded	8	54	36	1	2	0
release	1	2	5	6	0	0
releasec	10	48	13	4	2	0
RelGarbage	4	4	8	1	1	0
RelocHeap	3	3	6	1	0	0
remap	5	16	15	2	1	0
Remove	2	2	2	1	2	1
Rename	4	6	5	1	3	2
ResetTrail	5	5	13	1	2	0
restore	26	212784	100	1	11	5
restorevars	2	2	3	3	1	0
rstrv	2	2	2	2	0	0
save	4	8	34	1	9	3
savev	2	2	2	2	0	0
savevars	2	2	7	2	1	0
sbrk	•	•	•	1	•	•
scan	4	8	9	2	1	0
See	7	24	16	1	4	0
Seeing	1	1	1	1	0	0
Seen	1	1	2	3	1	0
setbuf	•	•	•	1	•	•
setjmp	•	•	•	1	•	•
SetPlPrompt	2	2	3	3	1	1
Sh	5	8	9	1	4	4
signal	•	•	•	3	•	•
sin	•	•	•	1	•	•
SortGarbage	6	8	12	1	0	0
sprintf	•	•	•	7	•	•
sqrt	•	•	•	1	•	•
sscanf	•	•	•	1	•	•
Stop	3	3	4	4	3	3
strcmp	•	•	•	7	•	•
strcpy	•	•	•	3	•	•
stringtolist	2	2	7	1	1	0
strlen	•	•	•	3	•	•
strncpy	•	•	•	1	•	•
SyntaxError	4	30	13	4	1	1
SysError	1	2	1	4	0	0
system	•	•	•	1	•	•
TakeSignal	7	10	16	1	6	2
tan	•	•	•	1	•	•
Tell	7	24	16	1	4	0
Telling	1	1	1	1	0	0
term	40	353600	84	4	8	0
times	•	•	•	1	•	•
ToEOL	6	45	8	2	0	0
token	35	130	103	4	6	1
Told	3	3	4	2	1	0
unifyarg	32	196992	56	3	2	0
unlink	•	•	•	2	•	•
Unwind	1	1	1	1	1	1

UserStartup	1	1	2	1	2	2
vvalue	3	4	6	6	1	0
wait	.	.	.	1	.	.
XtrFloat	1	1	3	4	0	0

Program: **Top**

Module	McCabe v(G)	NPATH	No. Statements	Fanin	Fanout	Fanout to Lib. Mods.
alarm	•	•	•	1	•	•
atoi	•	•	•	3	•	•
atoiwi	6	6	8	1	3	3
bcmp	•	•	•	1	•	•
bcopy	•	•	•	4	•	•
bzero	•	•	•	3	•	•
clear	2	3	2	1	2	1
ctime	•	•	•	1	•	•
end_screen	3	18	6	3	2	2
enter_user	4	6	11	1	3	2
error_count	1	1	1	1	0	0
err_compar	2	2	3	1	1	1
err_string	7	36	21	2	5	2
exit	•	•	•	4	•	•
exp	•	•	•	1	•	•
fflush	•	•	•	3	•	•
fmt_proc	1	2	3	2	4	2
fprintf	•	•	•	4	•	•
fputs	•	•	•	5	•	•
getenv	•	•	•	1	•	•
getkval	3	3	6	1	3	2
getopt	15	216	39	1	4	4
getpwent	•	•	•	1	•	•
getu	3	4	6	1	2	2
get_ucpu	5	5	9	1	2	1
get_user	3	3	5	1	3	1
index	•	•	•	2	•	•
init_hash	2	2	2	1	0	0
init_kernel	2	2	3	1	3	3
init_screen	2	3	10	1	2	2
init_termcap	11	576	30	1	9	9
ioctl	•	•	•	4	•	•
itoa	3	3	8	1	0	0
itoa7	4	6	10	2	0	0
i_cpustates	2	3	4	1	1	1
i_header	1	1	1	1	1	1
i_loadave	2	3	4	1	1	1
i_memory	1	1	1	1	1	1
i_process	1	7	6	1	4	3
i_procstates	3	7	6	1	2	2
kill	•	•	•	3	•	•
kill_procs	14	114	29	1	5	2
kvm_getu	•	•	•	1	•	•
kvm_open	•	•	•	1	•	•
kvm_read	•	•	•	1	•	•
leave	1	1	2	1	2	1
log	•	•	•	1	•	•
longjmp	•	•	•	1	•	•
main	81	-1466570000	272	0	58	28
next_field	3	4	6	2	1	1
nlist	•	•	•	1	•	•
onalrm	1	1	1	1	0	0
pause	•	•	•	1	•	•
perror	•	•	•	1	•	•
printable	3	3	6	1	0	0
printf	•	•	•	14	•	•
proc_compar	6	6	9	1	0	0
putstdout	1	7	1	3	0	0
qsort	•	•	•	2	•	•
quit	1	1	2	3	2	1
read	•	•	•	2	•	•
readline	10	212	20	1	4	4
reinit_screen	3	6	4	1	2	2
renice_procs	12	192	24	1	4	1
reset_display	1	1	6	1	6	0
rindex	•	•	•	1	•	•
sbrk	•	•	•	1	•	•
scanint	4	4	8	2	0	0
select	•	•	•	1	•	•
setbuffer	•	•	•	1	•	•
setjmp	•	•	•	1	•	•
setpriority	•	•	•	1	•	•
show_errors	2	6	4	1	1	1
show_help	1	1	2	1	1	1
sigblock	•	•	•	2	•	•
signal	•	•	•	2	•	•
sigsetmask	•	•	•	2	•	•

sleep	•	•	•	1	•	•
sprintf	•	•	•	1	•	•
standout	2	5	5	1	2	2
strcat	•	•	•	3	•	•
strcmp	•	•	•	3	•	•
strcpy	•	•	•	2	•	•
strlen	•	•	•	5	•	•
strncmp	•	•	•	1	•	•
strncpy	•	•	•	1	•	•
str_addarg	4	8	9	1	2	2
str_adderr	2	4	7	1	2	2
tgetent	•	•	•	1	•	•
tgetflag	•	•	•	1	•	•
tgetnum	•	•	•	1	•	•
tgetstr	•	•	•	1	•	•
tgoto	•	•	•	9	•	•
time	•	•	•	1	•	•
tputs	•	•	•	13	•	•
tstop	1	1	8	1	9	6
username	3	3	7	1	1	0
user_name	1	1	1	0	1	0
user_uid	1	1	1	0	1	0
u_cpustates	2	2	3	1	3	3
u_endscreen	6	20822	17	1	4	3
u_header	1	1	1	1	2	2
u_loadave	3	6	8	1	3	3
u_memory	1	1	6	1	3	3
u_process	10	4332	31	1	5	4
u_procstates	5	42	12	1	5	5
write	•	•	•	1	•	•
z_cpustates	2	3	4	1	1	1

Program: VN

Module	McCabe v(G)	NPATH	No. Statements	Fanin	Fanout	Fanout to Lib. Mods.
access	•	•	•	1	•	•
arg_opt	9	26	31	1	9	6
art_active	3	3	4	1	2	0
art_xfer	6	12	16	2	7	5
atoi	•	•	•	10	•	•
a_open	1	1	3	2	2	2
chdir	•	•	•	5	•	•
chkgroup	13	131	25	2	6	0
count_msg	6	8	10	1	1	0
creat	•	•	•	1	•	•
ctime	•	•	•	1	•	•
ctl_xlt	3	3	4	1	0	0
digclose	1	1	1	1	1	1
digest_extract	17	10368	53	2	16	9
digest_page	7	36	20	2	6	2
digname	21	7752	46	1	9	6
dig_advance	15	113	41	2	7	7
dig_list	4	6	11	2	7	4
dig_ulist	2	2	2	2	2	1
do_opt	5	4	13	1	2	0
do_out	16	50	51	1	2	2
do_update	5	9	9	2	2	0
do_write	3	4	5	2	2	1
edcopy	3	3	7	2	4	4
emptyline	3	4	5	1	0	0
envir_set	6	64	26	1	10	5
exit	•	•	•	1	•	•
fclose	•	•	•	14	•	•
fflush	•	•	•	7	•	•
fgets	•	•	•	12	•	•
fgprintf	2	2	3	4	2	2
fill_active	11	272	29	1	10	6
findall	4	4	4	1	1	1
find_page	7	32	20	4	4	2
first_ch	3	3	3	1	1	1
followup	8	40	30	1	16	8
fopen	•	•	•	13	•	•
form_title	3	3	8	2	4	3
forward	13	378	27	2	2	0
fprintf	•	•	•	10	•	•
fputs	•	•	•	3	•	•
free	•	•	•	2	•	•
fseek	•	•	•	6	•	•
ftell	•	•	•	6	•	•
fw_art	2	2	8	1	3	1
fw_chg	4	8	8	2	0	0
fw_done	3	3	3	1	1	0
fw_flush	1	1	5	3	1	0
fw_group	5	12	14	1	4	0
genlist	3	3	4	1	2	2
getenv	•	•	•	1	•	•
getkey	15	1110	28	1	1	1
getnoctl	8	16	6	6	0	0
getpgch	14	15	47	1	9	1
getpwnam	•	•	•	1	•	•
getpwuid	•	•	•	2	•	•
getuid	•	•	•	2	•	•
getwd	•	•	•	1	•	•
grp_indic	2	2	4	1	1	0
g_dir	2	2	3	2	3	3
hash	2	2	3	2	0	0
hashenter	3	4	19	2	5	0
hashfind	3	3	3	5	2	1
hashinit	2	2	3	1	0	0
help	8	146	22	1	5	2
help_rd	1	1	15	1	2	1
h_print	15	26	38	2	2	2
index	•	•	•	14	•	•
ioctl	•	•	•	1	•	•
last_ch	3	3	3	1	1	1
link	•	•	•	1	•	•
longjmp	•	•	•	1	•	•
lseek	•	•	•	2	•	•
mail	9	80	31	1	15	8
mail_cmd	4	6	12	1	2	1
mail_trim	4	6	9	1	3	2
main	2	4	19	0	16	1

make_newsorder	9	60	12	1	2	1
malloc	•	•	•	8	•	•
mkdir	•	•	•	1	•	•
mktemp	•	•	•	2	•	•
myfind	2	2	4	3	2	0
newsrc_opt	11	15	26	1	4	2
new_groups	9	320	21	2	4	3
new_read	2	2	3	2	0	0
new_sub	5	5	5	2	0	0
nfgets	19	3025	48	1	1	1
node_store	3	3	8	1	2	1
noslash	7	27	14	1	6	5
open	•	•	•	1	•	•
outc	1	7	1	1	0	0
outgroup	6	16	15	1	5	1
page_alloc	2	2	3	1	2	1
preinfo	2	14	11	6	5	4
prinfo	2	2	8	9	5	4
printex	2	2	11	26	7	3
printf	•	•	•	24	•	•
printr	3	4	16	1	14	11
printstr	6	20	17	1	10	4
read	•	•	•	1	•	•
readfile	37	187938	133	1	22	11
readstr	11	111	21	1	7	1
regcmp	5	10	16	4	4	2
regex	7	36	14	4	3	2
regfree	4	4	10	4	0	0
rerrmsg	1	1	5	6	2	1
rewind	•	•	•	3	•	•
re_comp	•	•	•	2	•	•
re_exec	•	•	•	1	•	•
rindex	•	•	•	2	•	•
rot_line	8	17	10	3	0	0
rprompt	1	1	3	2	3	2
saver	4	6	6	1	5	1
savestr	7	24	23	1	13	3
save_art	15	3456	42	2	12	7
save_article	4	280	14	1	6	6
searcher	10	168	29	1	11	5
session	76	84888	294	1	35	5
setjmp	•	•	•	1	•	•
set_kxln	11	44	26	1	6	5
show	10	180	26	3	4	2
sigcatch	10	36	24	1	8	4
signal	•	•	•	2	•	•
sig_set	12	22	31	3	4	1
specfilter	6	16	17	1	2	0
specmark	12	342	28	1	8	2
spec_group	6	18	15	1	7	0
sprintf	•	•	•	32	•	•
srch_help	4	6	7	1	0	0
stat	•	•	•	2	•	•
stat_end	1	0	0	2	0	0
stat_start	1	0	0	1	0	0
strcmp	•	•	•	5	•	•
strcpy	•	•	•	20	•	•
strlen	•	•	•	20	•	•
strncmp	•	•	•	7	•	•
strpbrk	4	6	6	1	1	1
strtok	4	8	11	9	2	0
str_store	5	10	13	9	4	3
str_taptr	1	1	2	0	0	0
str_tfree	2	2	4	2	1	1
str_tpool	2	2	7	2	2	1
str_tstore	3	4	10	4	4	3
system	•	•	•	6	•	•
temp_open	2	2	6	1	4	2
term_set	21	242	55	21	7	3
tgetent	•	•	•	1	•	•
tgetflag	•	•	•	1	•	•
tgetnum	•	•	•	1	•	•
tgetstr	•	•	•	1	•	•
tgoto	•	•	•	1	•	•
time	•	•	•	1	•	•
tmpnam	2	2	5	7	3	3
tot_list	15	550	34	1	8	4
tputs	•	•	•	2	•	•
tty_set	13	24	42	9	3	1
twiddle	6	12	14	2	3	3

t_setup	15	1296	42	1	10	5
ungetc	•	•	•	1	•	•
unlink	•	•	•	10	•	•
up_seen	5	7	6	1	1	0
userlist	20	1170	40	1	11	5
user_str	15	40864	40	8	5	3
vns_aclose	2	2	4	3	2	1
vns_aopen	29	5313020	102	3	13	8
vns_asave	2	2	4	1	4	3
vns_envir	2	2	12	1	5	3
vns_exit	1	0	0	2	0	0
vns_gset	2	2	3	4	3	1
vns_news	29	1259710	64	1	19	8
vns_write	11	76	20	2	6	5
vn_env	5	8	7	4	2	2
write	•	•	•	1	•	•
write_page	2	2	5	1	2	1
wr_show	3	3	8	1	2	1
xln_str	3	3	3	1	1	1

Program: **WM**

Module	McCabe v(G)	NPATH	No. Statements	Fanin	Fanout	Fanout to Lib. Mods.
abs	•	•	•	1	•	•
access	•	•	•	1	•	•
add_to_try	16	432	51	1	5	3
askwindow	11	11	14	1	2	0
baudrate	3	4	5	1	0	0
bcopy	•	•	•	1	•	•
beep	1	7	1	2	0	0
ClearScreen	1	1	2	3	2	2
close	•	•	•	3	•	•
covers	6	6	5	1	0	0
delwin	•	•	•	2	•	•
DisablePty	2	2	2	3	0	0
docmd	63	146	160	1	23	1
DoCmdArgs	6	7	16	1	3	3
DumpWindow	6	982	15	1	3	3
dup2	•	•	•	1	•	•
EnablePty	2	2	2	2	0	0
endwin	•	•	•	3	•	•
erasechar	1	1	1	2	0	0
execlp	•	•	•	2	•	•
exit	•	•	•	4	•	•
fclose	•	•	•	3	•	•
fflush	•	•	•	4	•	•
fitwindow	12	336	26	1	3	1
flash	2	2	3	4	2	1
fopen	•	•	•	3	•	•
fork	•	•	•	1	•	•
fprintf	•	•	•	4	•	•
free	•	•	•	2	•	•
FreeWindow	6	32	16	5	1	1
fscanf	•	•	•	1	•	•
getbounds	9	64	34	1	8	2
getcap	•	•	•	3	•	•
getenv	•	•	•	2	•	•
getpgrp	•	•	•	1	•	•
getpid	•	•	•	1	•	•
getpos	23	43522	78	1	8	1
getppid	•	•	•	1	•	•
GetSlot	3	3	4	2	0	0
gtty	•	•	•	1	•	•
helpmsg	1	1	24	1	3	2
IdentWindows	12	514	27	1	7	1
InitPseudoTty	1	1	5	1	1	1
initscr	•	•	•	1	•	•
init_keypad	3	4	5	1	4	3
ioctl	•	•	•	9	•	•
kill	•	•	•	2	•	•
killchar	1	1	1	2	0	0
killpg	•	•	•	1	•	•
KillShell	2	2	5	3	2	2
longjmp	•	•	•	1	•	•
main	28	453726	69	0	30	7
MakeBorders	15	2500	42	1	3	3
Malloc	1	1	1	4	1	1
malloc	•	•	•	1	•	•
mkprint	1	12	4	7	0	0
movecursor	10	90	14	7	2	2
mvcur	•	•	•	1	•	•
mvwin	•	•	•	1	•	•
namecmp	4	6	5	1	0	0
NewShell	10	32	39	1	13	8
newwin	•	•	•	4	•	•
NewWindow	9	72	22	5	7	3
onintr	1	1	2	1	2	1
open	•	•	•	3	•	•
overlap	1	3	3	1	0	0
overwrite	9	400	32	5	0	0
plural	1	2	1	1	0	0
printf	•	•	•	1	•	•
read	•	•	•	2	•	•
readptys	7	25	16	1	6	2
ReapShell	9	31	16	1	8	4
RedrawScreen	12	330	21	4	5	3
Restore	28	870944	59	1	10	6
RestoreCursor	2	2	3	5	2	1
RestoreMsg	2	2	3	3	2	2
rindex	•	•	•	1	•	•

Save	4	20	10	2	4	3
select	•	•	•	1	•	•
setbuf	•	•	•	1	•	•
SetCntrlTerm	1	1	4	1	3	3
setenv	9	36	17	1	5	3
setitimer	•	•	•	1	•	•
setjmp	•	•	•	1	•	•
setpgrp	•	•	•	1	•	•
SetProcGrp	1	1	3	1	3	3
SetTerm	14	66	25	1	6	3
ShellInit	2	8	11	1	5	5
showmsg	5	16	20	14	11	9
Shutdown	3	21	8	3	5	2
sigalrm	1	1	1	1	1	1
sigchild	1	1	2	1	2	1
signal	•	•	•	4	•	•
sleep	•	•	•	1	•	•
sprintf	•	•	•	8	•	•
Startup	22	57600	44	1	15	12
strcat	•	•	•	2	•	•
strcmp	•	•	•	2	•	•
strcpy	•	•	•	8	•	•
strlen	•	•	•	8	•	•
suspend	6	112	21	1	13	11
system	•	•	•	1	•	•
termcap	13	480	15	2	3	3
tgoto	•	•	•	2	•	•
time	•	•	•	2	•	•
touchwin	•	•	•	4	•	•
tparm	81	365	234	2	3	3
tputs	•	•	•	7	•	•
tstp	•	•	•	1	•	•
tty_getch	13	488	32	6	6	3
tty_inputpending	4	6	5	1	1	1
tty_realgetch	3	3	11	1	1	1
ualarm	2	2	6	1	1	1
untouchwin	2	2	2	2	0	0
waddch	•	•	•	3	•	•
waddstr	•	•	•	2	•	•
wait3	•	•	•	1	•	•
wclear	•	•	•	1	•	•
wclrtobot	•	•	•	1	•	•
wclrtoeol	•	•	•	2	•	•
wdelch	•	•	•	1	•	•
wdeleteln	•	•	•	2	•	•
werase	•	•	•	3	•	•
winchanged	6	20	16	1	2	0
winsch	•	•	•	1	•	•
winsertln	•	•	•	2	•	•
WListAdd	3	3	8	3	1	0
WListDelete	10	84	15	2	2	0
WMaddbuf	26	69	57	3	8	3
WMdeleteln	14	36	55	2	12	6
WMescape	36	360	89	1	17	10
WMinsertln	6	6	35	1	11	6
wmove	•	•	•	7	•	•
WObscure	5	7	10	3	2	0
WPrompt	15	120	26	2	11	6
wrefresh	•	•	•	11	•	•
write	•	•	•	2	•	•
wstandend	•	•	•	1	•	•
wstandout	•	•	•	3	•	•
ZapMsgLine	2	2	3	1	2	2

Program: Xscheme

Module	McCabe v(G)	NPATH	No. Statements	Fanin	Fanout	Fanout to Lib. Mods.
acos	•	•	•	1	•	•
addivar	1	1	1	1	2	0
addmsg	1	1	2	6	2	0
add_level	1	1	5	1	2	0
allocnode	3	3	9	5	2	0
allocvector	7	36	25	6	4	0
asin	•	•	•	1	•	•
assoc	8	308	11	3	3	0
atan	•	•	•	2	•	•
atan2	•	•	•	1	•	•
atof	•	•	•	1	•	•
atol	•	•	•	1	•	•
badargtype	1	1	1	1	1	0
badfop	1	1	1	3	1	0
badfuntype	1	1	1	1	1	0
badiop	1	1	1	3	1	0
binary	52	162480	137	12	9	1
callerrorhandler	3	3	11	2	5	1
calloc	•	•	•	3	•	•
cd_evariable	1	1	3	5	1	0
cd_fundefinition	2	2	10	2	8	0
cd_let	10	216	22	2	14	0
cd_literal	3	3	6	1	2	0
cd_variable	1	1	2	5	2	0
ceil	•	•	•	1	•	•
checkeof	2	2	3	2	2	0
checkfneg	2	2	2	1	1	0
checkfzero	2	2	2	1	1	0
checkineg	2	2	2	1	1	0
checkizero	2	2	2	1	1	0
chrcompare	10	4900	23	10	3	0
clanswer	3	2744	17	1	8	0
clisnew	4	31752	27	1	11	0
clnew	1	7	3	1	5	0
compact	4	8	7	1	1	0
compact_vector	6	9	17	1	0	0
compare	27	10480	69	5	2	0
cons	4	5	16	37	3	0
copylists	8	72	18	1	2	0
cos	•	•	•	1	•	•
curinput	1	1	1	6	0	0
curoutput	1	1	1	12	0	0
cvchar	1	1	3	6	1	0
cvclosure	1	1	3	4	1	0
cvfixnum	3	3	5	26	1	0
cvflonum	1	1	3	4	1	0
cviptr	10	84	14	1	2	0
cvmethod	1	1	3	1	1	0
cvoptr	6	12	7	1	1	0
cvport	1	1	5	4	1	0
cvpromise	1	1	3	1	2	0
cvstring	1	1	3	4	3	2
cvsubr	1	1	4	1	1	0
cvsymbol	2	2	9	3	3	0
cxr	7	294	9	28	3	0
decode_instruction	14	72	54	2	4	1
decode_procedure	2	2	5	1	1	0
define1	12	105	21	2	11	0
do_access	7	45	9	1	4	0
do_and	6	10	10	1	5	0
do_begin	7	13	7	9	3	0
do_call	3	4	9	2	5	0
do_cond	9	28	17	1	7	0
do_continuation	3	2	6	14	1	0
do_define	3	5	3	1	2	0
do_delay	4	10	14	1	9	0
do_expr	9	15	7	16	5	0
do_forloop	6	14	24	2	4	0
do_identifier	3	3	6	1	5	0
do_if	9	300	22	1	6	0
do_lambda	3	5	4	1	3	0
do_let	5	9	3	1	2	0
do_letrec	8	72	21	1	13	0
do_letstar	9	72	10	1	6	0
do_literal	1	1	2	9	2	0
do_load	3	28	12	2	7	0
do_loadloop	4	5	19	2	5	0

do_maploop	8	28	30	2	4	0
do_named_let	4	8	19	1	10	0
do_nary	3	3	5	1	5	0
do_or	6	10	10	1	5	0
do_quote	3	5	3	1	2	0
do_set	7	13	7	1	3	0
do_setaccess	12	1350	18	1	6	0
do_setvar	4	10	9	1	6	0
do_while	3	5	12	1	7	0
do_withfile	5	224	21	2	8	0
entermsg	4	6	9	2	2	0
eq	1	1	1	1	0	0
eqtest	2	16	6	3	2	0
equal	9	64	17	4	3	1
eqv	6	38	11	1	0	0
errprint	1	1	2	3	2	0
errputstr	1	1	1	3	1	0
exit	•	•	•	3	•	•
exp	•	•	•	1	•	•
fclose	•	•	•	1	•	•
fgets	•	•	•	1	•	•
findcvariable	4	6	6	2	2	0
findliteral	5	7	7	3	2	0
findmemory	4	8	11	2	0	0
findprop	6	10	4	1	0	0
findvar	4	4	7	4	0	0
findvariable	6	10	7	2	2	0
findvmemory	6	12	15	2	2	0
find_internal_definitions	13	43	15	1	0	0
fixup	2	2	6	10	0	0
floor	•	•	•	2	•	•
fopen	•	•	•	2	•	•
free	•	•	•	1	•	•
freeimage	9	18	15	1	2	1
fseek	•	•	•	1	•	•
ftell	•	•	•	1	•	•
gc	18	8748	20	5	4	0
gc_protect	2	2	5	1	0	0
getdigit	1	2	1	1	0	0
getivcnt	4	12	3	3	1	0
getsymbol	4	5	6	2	3	0
getvspace	3	3	9	1	2	0
hash	2	4	4	2	0	0
info	2	2	3	1	2	1
in_ftab	3	3	6	1	2	1
in_ntab	3	3	6	1	1	1
isnumber	18	5832	27	1	5	3
isradixdigit	5	6	9	1	0	0
issym	4	4	6	2	0	0
length	3	5	3	5	0	0
letstar1	4	10	17	2	11	0
listlength	3	5	3	1	0	0
log	•	•	•	1	•	•
longjmp	•	•	•	5	•	•
main	4	24	13	1	10	4
makearray1	6	80	13	1	2	0
make_code_object	5	20	13	2	4	0
make_continuation	3	4	11	8	4	0
mark	23	97	46	2	1	0
markvector	5	5	5	1	1	0
member	5	168	10	3	3	0
newcode	1	1	1	2	1	0
newcontinuation	1	1	1	2	1	0
newframe	1	1	3	4	2	0
newnsegment	3	4	13	2	1	1
newobject	1	1	3	2	1	0
newstring	1	1	3	6	1	0
newvector	1	1	1	8	1	0
newvsegment	2	2	9	1	1	1
nth	7	28224	12	2	4	0
obclass	2	14	5	1	3	0
obisnew	2	14	5	1	3	0
obshow	3	280	21	1	7	0
obsymbols	1	1	4	1	1	0
openfile	6	2016	16	4	8	0
osagetc	1	2	1	1	0	0
osaopen	1	1	1	4	1	1
osaputc	1	7	1	3	0	0
osbgetc	1	2	1	4	0	0
osbopen	1	1	1	3	1	1

osbputc	1	7	1	5	0	0
oscheck	1	0	0	4	0	0
osclose	1	1	1	12	1	1
oserror	1	1	1	1	1	1
osfinish	1	0	0	1	0	0
osflush	1	1	1	1	0	0
osinit	1	1	3	1	1	1
osrand	3	4	6	1	0	0
osseek	1	1	1	1	1	1
ossymbols	1	0	0	1	0	0
ostell	1	1	1	1	1	1
ostgetc	5	12	10	1	3	2
ostputc	2	14	4	2	2	0
parse_lambda_list	37	230384	58	2	4	0
parse_let_variables	11	31	20	3	4	0
pow	•	•	•	1	•	•
predicate	16	184	40	5	5	0
print	35	168	89	3	12	0
printf	•	•	•	2	•	•
push_args	3	5	6	2	3	0
push_dummy_values	5	9	5	1	1	0
push_init_expressions	7	13	8	3	3	0
push_nargs	6	16	10	2	4	0
putatm	1	1	5	2	3	1
putcbyte	2	2	5	29	1	0
putcharacter	4	3	11	1	2	1
putclosure	1	1	1	1	1	0
putcode	2	2	4	2	3	1
putconstant	1	1	2	1	2	0
putcword	1	1	3	10	1	0
putflonum	1	5	2	1	2	1
putnumber	1	5	2	1	2	1
putoct	1	1	2	1	2	1
putstring	12	11	26	1	2	0
putsubr	1	1	2	1	2	1
putsym	5	12	8	2	2	0
readnode	2	2	3	1	4	0
readptr	2	2	3	1	8	0
read_cdr	4	8	7	1	1	0
read_comma	2	2	4	1	1	0
read_comment	4	6	4	1	4	0
read_list	10	22	28	1	3	0
read_quote	3	4	8	4	2	0
read_radix	5	6	8	1	9	0
read_special	24	23	49	2	5	0
read_string	4	5	7	1	8	0
read_symbol	2	4	3	1	14	3
read_vector	9	16	27	1	2	0
remove_level	1	1	4	6	0	0
restore_continuation	2	2	3	1	0	0
scan	3	3	3	4	1	0
setit	3	58	7	3	0	0
setjmp	•	•	•	1	•	•
setoffset	2	2	4	1	4	0
set_bound_variables	8	14	6	2	3	0
sin	•	•	•	1	•	•
sprintf	•	•	•	11	•	•
sqrt	•	•	•	1	•	•
stdprint	1	1	2	1	2	0
stdputstr	1	1	1	2	1	0
strcat	•	•	•	1	•	•
strcmp	•	•	•	5	•	•
strcompare	18	66248	42	10	3	0
strcpy	•	•	•	1	•	•
strlen	•	•	•	4	•	•
strncpy	•	•	•	1	•	•
sweep	2	2	4	1	1	0
sweep_segment	5	4	12	1	1	0
system	•	•	•	1	•	•
tan	•	•	•	1	•	•
toflotype	3	5	5	1	0	0
tolower	•	•	•	3	•	•
toupper	•	•	•	3	•	•
unary	34	212	88	13	19	9
vectorequal	4	6	6	1	1	0
vref	4	70	7	2	4	0
vset	4	140	9	2	4	0
wrapup	2	2	4	2	3	1
writenode	2	2	3	1	1	0
writeptr	2	2	2	2	1	0

xabs	1	1	1	1	1	0
xacos	1	1	1	1	1	0
xadd	1	1	1	2	1	0
xadd1	1	1	1	1	1	0
xappend	8	70	14	1	3	0
xapply	5	280	12	1	6	0
xaref	7	2954	15	1	5	0
xaset	7	2954	15	1	5	0
xasin	1	1	1	1	1	0
xassoc	1	1	1	1	1	0
xassq	1	1	1	1	1	0
xassv	1	1	1	1	1	0
xatan	3	210	9	1	7	2
xatom	2	20	5	1	2	0
xbooleanp	2	12	5	1	2	0
xboundp	2	28	5	1	3	0
xcaaaar	1	1	1	1	1	0
xcaaadr	1	1	1	1	1	0
xcaaar	1	1	1	1	1	0
xcaadar	1	1	1	1	1	0
xcaaddr	1	1	1	1	1	0
xcaadr	1	1	1	1	1	0
xcaar	1	1	1	1	1	0
xcadaar	1	1	1	1	1	0
xcadadr	1	1	1	1	1	0
xcadar	1	1	1	1	1	0
xcaddar	1	1	1	1	1	0
xcadddr	1	1	1	1	1	0
xcaddr	1	1	1	1	1	0
xcadr	1	1	1	1	1	0
xcallcc	4	16	14	1	5	0
xcallwi	1	1	1	1	1	0
xcallwo	1	1	1	1	1	0
xcar	2	28	5	1	3	0
xcdaaar	1	1	1	1	1	0
xcdaadr	1	1	1	1	1	0
xcdaar	1	1	1	1	1	0
xcdadar	1	1	1	1	1	0
xcdaddr	1	1	1	1	1	0
xcdadr	1	1	1	1	1	0
xcdar	1	1	1	1	1	0
xcddaar	1	1	1	1	1	0
xcddadr	1	1	1	1	1	0
xcddar	1	1	1	1	1	0
xcdddar	1	1	1	1	1	0
xcddddr	1	1	1	1	1	0
xcdddr	1	1	1	1	1	0
xcddr	1	1	1	1	1	0
xcdr	2	28	5	1	3	0
xceiling	6	36	9	1	5	1
xcharint	2	14	5	1	4	0
xcharp	2	20	5	1	2	0
xchreql	1	1	1	1	1	0
xchrgeq	1	1	1	1	1	0
xchrgtr	1	1	1	1	1	0
xchrieql	1	1	1	1	1	0
xchrigeq	1	1	1	1	1	0
xchrigtr	1	1	1	1	1	0
xchrileq	1	1	1	1	1	0
xchrilss	1	1	1	1	1	0
xchrleq	1	1	1	1	1	0
xchrlss	1	1	1	1	1	0
xclose	3	28	8	1	4	0
xclosei	3	32	8	1	4	0
xcloseo	3	32	8	1	4	0
xcompile	3	72	13	1	6	0
xcons	2	8	6	1	3	0
xcos	1	1	1	1	1	0
xcurinput	2	2	4	1	2	0
xcuroutput	2	2	4	1	2	0
xdecompile	6	360	9	1	5	0
xdefaultobjectp	2	8	5	1	2	0
xdisplay	2	40	7	1	5	0
xdiv	1	1	1	1	1	0
xenvbindings	9	216	23	1	5	0
xenvp	2	20	5	1	2	0
xenvparent	2	14	5	1	3	0
xeofobjectp	2	8	5	1	2	0
xeq	1	1	1	1	2	0
xeql	1	1	1	2	1	0

xequal	1	1	1	1	2	0
xeqv	1	1	1	1	2	0
xerror	2	21	10	1	5	0
xevenp	1	1	1	1	1	0
xexactp	2	20	5	1	3	0
xexit	2	2	4	1	2	0
xexp	1	1	1	1	1	0
xexpt	1	1	1	1	1	0
xfloor	6	36	9	1	5	1
xforce	6	28	18	1	5	0
xforce1	1	1	4	1	1	0
xforeach	2	2	3	1	2	0
xforeach1	1	2	3	1	1	0
xgc	3	5	9	1	4	0
xgcd	7	4760	18	1	3	0
xgensym	7	30	19	1	6	2
xgeq	1	1	1	1	1	0
xget	2	98	6	1	4	0
xgetfposition	2	14	5	1	5	0
xgtr	1	1	1	2	1	0
xicar	2	4	5	1	2	0
xicdr	2	4	5	1	2	0
xinexactp	2	20	5	1	3	0
xinputportp	2	24	5	1	2	0
xintchar	2	28	5	1	4	0
xintegerp	2	20	5	1	2	0
xisetcar	2	8	7	1	2	0
xisetcdr	2	8	7	1	2	0
xivlength	2	4	5	1	3	0
xivref	1	2	1	1	2	0
xivset	1	2	1	1	2	0
xlabort	1	1	7	5	4	1
xlapply	10	36	34	11	8	0
xlastpair	6	126	8	1	3	0
xlbadtype	1	1	1	96	1	0
xlbreak	1	1	1	1	1	0
xlcleanup	1	0	0	1	0	0
xlcompile	2	2	13	1	6	0
xlcontinue	1	0	0	1	0	0
xlength	4	70	7	1	4	0
xlenter	3	3	9	20	4	1
xleq	1	1	1	1	1	0
xlerror	2	2	9	40	3	0
xlexecute	104	789	249	1	24	1
xlfail	1	1	1	17	1	0
xlfatal	1	1	2	3	2	1
xlflush	1	1	1	1	1	0
xlfunction	2	2	12	1	6	0
xlgetc	6	6	11	12	3	0
xlgetprop	1	2	1	1	1	0
xlinitws	6	80	142	1	16	2
xlirestore	22	40	69	2	11	0
xlisave	19	38	57	1	8	0
xlist	3	3	11	1	1	0
xlistp	2	20	5	1	2	0
xlistref	1	1	1	1	1	0
xliststring	5	84	12	1	5	0
xlisttail	1	1	1	1	1	0
xlistvect	3	28	9	1	5	0
xlload	1	0	0	1	0	0
xlminit	2	2	15	2	2	1
xload	1	1	1	1	1	0
xload1	2	2	5	1	4	0
xloadnoisily	1	1	1	1	1	0
xlog	1	1	1	1	1	0
xlogand	1	1	1	1	1	0
xlogior	1	1	1	1	1	0
xlognot	1	1	1	1	1	0
xlogxor	1	1	1	1	1	0
xloinit	1	1	28	1	6	0
xlpeek	2	2	5	1	1	0
xlprin1	1	1	1	7	1	0
xlprinc	1	1	1	1	1	0
xlputc	5	5	9	10	3	0
xlputprop	2	2	3	1	2	0
xlputstr	2	2	2	14	1	0
xlread	11	10	30	6	10	0
xlreturn	4	5	14	9	1	0
xlsend	5	5	9	2	2	0
xlss	1	1	1	2	1	0

xlstkover	1	1	1	39	1	0
xlsubr	1	1	2	1	2	0
xlsymbols	1	1	28	2	3	0
xlterpri	1	1	1	7	1	0
xltoofew	1	1	1	131	1	0
xltoomany	1	1	1	121	1	0
xltoplevel	1	1	2	1	2	1
xlungetc	2	2	3	8	0	0
xmakearray	1	1	3	1	1	0
xmakevector	5	238	17	1	5	0
xmap	2	2	4	1	2	0
xmap1	2	4	8	1	2	0
xmax	1	1	1	1	1	0
xmember	1	1	1	1	1	0
xmemq	1	1	1	1	1	0
xmemv	1	1	1	1	1	0
xmin	1	1	1	1	1	0
xmul	1	1	1	2	1	0
xnegativep	1	1	1	1	1	0
xnewline	2	20	6	1	5	0
xnull	2	8	5	1	2	0
xnumberp	2	32	5	1	2	0
xobjectp	2	20	5	1	2	0
xoddp	1	1	1	1	1	0
xopena	1	1	1	1	1	0
xopeni	1	1	1	1	1	0
xopeno	1	1	1	1	1	0
xopenu	1	1	1	1	1	0
xoutputportp	2	24	5	1	2	0
xpairp	2	20	5	1	2	0
xportp	2	20	5	1	2	0
xpositivep	1	1	1	1	1	0
xprbreadth	1	1	1	1	1	0
xprdepth	1	1	1	1	1	0
xprint	2	40	8	1	6	0
xprocedurep	2	36	5	1	2	0
xprocenvironment	2	14	5	1	3	0
xput	2	196	8	1	4	0
xquo	1	1	1	1	1	0
xrandom	1	1	1	1	1	0
xrdbyte	2	40	5	1	6	0
xrdchar	2	40	5	1	6	0
xrdlong	4	60	9	1	6	0
xrdshort	4	60	9	1	6	0
xread	3	40	7	1	5	0
xrealp	2	20	5	1	2	0
xrem	1	1	1	1	1	0
xreset	2	2	4	1	2	1
xrestore	3	28	8	1	6	1
xreverse	4	70	9	1	4	0
xround	9	48	18	1	5	1
xsave	2	28	5	1	4	0
xsendsuper	7	175	17	1	4	0
xsetcar	2	28	7	1	3	0
xsetcdr	2	28	7	1	3	0
xsetfposition	2	5488	9	1	4	0
xsetsymplist	2	28	7	1	3	0
xsetsymvalue	2	28	7	1	3	0
xsin	1	1	1	1	1	0
xsqrt	1	1	1	1	1	0
xstrappend	3	16	13	1	4	1
xstreql	1	1	1	1	1	0
xstrgeq	1	1	1	1	1	0
xstrgtr	1	1	1	1	1	0
xstrieql	1	1	1	1	1	0
xstrigeq	1	1	1	1	1	0
xstrigtr	1	1	1	1	1	0
xstrileq	1	1	1	1	1	0
xstrilss	1	1	1	1	1	0
xstringp	2	20	5	1	2	0
xstrlen	2	14	5	1	4	0
xstrleq	1	1	1	1	1	0
xstrlist	4	56	13	1	6	0
xstrlss	1	1	1	1	1	0
xstrmullp	2	28	5	1	3	0
xstrref	4	490	8	1	5	0
xstrsym	2	14	5	1	4	0
xsub	1	1	1	2	1	0
xsub1	1	1	1	1	1	0
xsubstring	8	50568	22	1	5	0

xsymbolp	2	20	5	1	2	0
xsymplist	2	14	5	1	3	0
xsymstr	2	14	5	1	3	0
xsymvalue	2	14	5	1	3	0
xsystem	3	32	6	1	5	1
xtan	1	1	1	1	1	0
xtheenvironment	2	2	4	1	1	0
xtraceoff	2	2	4	1	1	0
xtraceon	2	2	4	1	1	0
xtransoff	3	4	8	1	2	0
xtranson	3	56	8	1	5	0
xtruncate	6	36	9	1	4	0
xvectlist	4	56	13	1	5	0
xvector	2	3	4	1	2	0
xvectorp	2	20	5	1	2	0
xvlength	2	14	5	1	4	0
xvref	1	7	1	2	3	0
xvset	1	7	1	2	3	0
xwithfile1	1	1	3	1	2	0
xwrbyte	2	280	7	1	5	0
xwrchar	2	140	7	1	5	0
xwrite	2	40	7	1	5	0
xwrlong	3	560	9	1	5	0
xwrshort	3	560	9	1	5	0
xzerop	1	1	1	1	1	0

Program: Yap

Module	McCabe v(G)	NPATH	No. Statements	Fanin	Fanout	Fanout to Lib. Mods.
addstring	3	3	3	2	1	0
addtolist	2	2	5	2	0	0
addtomach	8	19	21	1	1	0
alloc	10	42	26	4	6	4
basename	6	14	10	1	0	0
bottom	2	2	3	2	3	1
catchdel	1	2	2	2	2	1
close	•	•	•	5	•	•
clrbline	3	4	7	8	2	1
cls	2	2	2	1	1	1
cls_files	1	1	3	1	1	0
compile	17	3169	41	1	7	3
compretval	6	12	7	2	1	0
cputline	9	31	15	3	1	0
display	13	360	36	11	9	1
do_absolute	3	3	5	3	3	0
do_backward	3	4	11	1	4	0
do_bscreens	4	4	4	1	2	0
do_bsearch	1	1	1	1	1	0
do_bskiplines	1	1	2	1	2	0
do_b_scroll	1	1	1	1	2	0
do_chkm	2	2	5	2	1	0
do_clean	5	12	18	2	1	1
do_comm	19	2176	33	2	2	0
do_error	1	1	1	1	1	0
do_firstline	1	1	1	1	1	0
do_forward	2	2	5	1	3	0
do_fscreens	4	4	4	1	2	0
do_fsearch	1	1	1	1	1	0
do_f_scroll	1	1	1	1	2	0
do_help	4	6	9	1	4	0
do_lcomm	2	2	4	1	2	0
do_lf	2	2	4	1	2	0
do_line	32	166342	66	4	5	1
do_lline	3	4	6	3	4	0
do_nextfile	2	2	4	1	33	0
do_nocomm	1	0	0	1	0	0
do_nsearch	1	1	1	1	1	0
do_previousfile	2	2	4	1	3	0
do_redraw	1	1	1	1	1	0
do_msearch	1	1	1	1	1	0
do_search	11	250	24	3	8	2
do_shell	4	4	9	1	5	0
do_skiplines	1	1	2	1	2	0
do_upline	2	2	4	1	2	0
do_visit	4	6	10	1	5	1
do_writefile	9	45	19	1	6	3
dup	•	•	•	2	•	•
d_clean	1	1	5	2	0	0
error	1	1	1	13	0	0
execl	•	•	•	1	•	•
execve	•	•	•	1	•	•
exgmark	1	1	5	1	1	0
exit	•	•	•	3	•	•
fillnum	2	2	5	2	1	0
fillscr	2	2	2	1	1	0
flush	2	2	4	10	1	1
fork	•	•	•	1	•	•
fputch	3	3	5	3	1	0
free	•	•	•	3	•	•
fstat	•	•	•	2	•	•
getblock	13	234	21	2	2	0
getcap	2	2	4	1	1	1
getch	12	54	14	3	5	3
getcomm	8	27	20	1	3	0
getenv	•	•	•	3	•	•
getline	2	2	3	8	1	0
getnum	1	1	2	2	1	0
getpid	•	•	•	1	•	•
getpos	1	1	4	1	1	0
give_prompt	28	8800	73	1	8	0
handle	2	2	2	1	1	0
home	1	1	1	3	1	1
initialize	7	36	21	1	9	7
initkeys	3	3	11	1	4	1
inittty	5	12	27	2	4	2
ini_terminal	34	79626200	91	1	15	9

ioctl	•	•	•	5	•	•
isatty	•	•	•	2	•	•
isused	1	1	1	2	0	0
kerror	1	1	3	1	2	2
killchar	5	9	11	2	3	0
lookup	4	6	10	1	1	1
lseek	•	•	•	2	•	•
main	6	24	18	0	10	4
malloc	•	•	•	1	•	•
match	7	32	15	2	0	0
mgoto	10	10	13	2	5	2
mktemp	•	•	•	1	•	•
new_block	6	7	17	2	2	0
nextblock	20	3150	67	1	5	1
nextfile	3	3	5	4	1	0
next_screen	3	4	6	2	2	0
nflush	1	1	2	1	0	0
n_or_m_search	4	6	11	2	5	0
open	•	•	•	4	•	•
opentemp	2	4	5	1	4	3
page_size	4	5	7	2	0	0
panic	1	1	4	4	2	0
parsopt	11	60	26	1	0	0
prev_screen	3	3	4	2	2	0
pmum	1	1	1	1	2	0
processfiles	6	24	23	1	8	1
pr_comm	6	14	14	1	1	0
pr_mach	7	25	18	2	4	0
putline	5	8	8	20	1	0
quit	1	1	3	4	3	1
read	•	•	•	3	•	•
readblock	3	4	12	1	5	2
readline	15	452	32	5	7	1
readoptions	7	24	12	1	3	1
redraw	3	3	4	13	1	0
resettty	2	2	7	2	3	2
ret_to_continue	4	4	11	3	5	0
re_alloc	4	5	10	2	2	1
re_comp	•	•	•	1	•	•
re_exec	•	•	•	1	•	•
scrollb	20	31752	44	6	7	1
scrollf	12	360	28	4	6	0
scro_size	3	4	5	2	0	0
setjmp	•	•	•	1	•	•
setmark	1	1	2	1	0	0
setpgrp	•	•	•	2	•	•
setused	1	1	1	1	0	0
shellescape	18	352	56	2	17	7
signal	•	•	•	3	•	•
skiplines	2	2	4	3	1	0
strcat	•	•	•	3	•	•
strcmp	•	•	•	2	•	•
strcpy	•	•	•	7	•	•
strlen	•	•	•	2	•	•
tgetent	•	•	•	1	•	•
tgetflag	•	•	•	1	•	•
tgetnum	•	•	•	1	•	•
tgetstr	•	•	•	1	•	•
tgoto	•	•	•	3	•	•
tomark	1	1	1	2	1	0
to_lastline	4	4	5	1	1	0
tputs	•	•	•	7	•	•
unlink	•	•	•	1	•	•
visitfile	3	4	10	2	6	3
wait	•	•	•	1	•	•
window	13	288	25	1	3	3
write	•	•	•	3	•	•

Appendix D3 - Basic Statistical Data for Metric Values by Program.

Bib

X_1 : Fanin

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.226	2.626	.272	6.894	117.965	93
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	15	15	207	1095	0
Mode:					
1					

X_2 : Fanout

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
3.339	3.854	.489	14.851	115.423	62
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	18	18	207	1597	0
Mode:					
2					

X_3 : Fanout to Lib. Mods.

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
1.823	2.399	.305	5.755	131.623	62
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	14	14	113	557	0
Mode:					
0					

X_4 : No. Statements

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
20.21	26.758	3.398	715.972	132.4	62
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	135	134	1253	68997	0
Mode:					
5					

X_5 : McCabe

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
9.177	10.675	1.356	113.952	116.316	62
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	55	54	569	12173	0
Mode:					
1					

X_6 : NPATH

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
344702.613	2364616.499	300306.596	5.591E12	685.987	62
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	18445700	18445699	21371562	3.484E14	0
Mode:					
1					

Bison

X_1 : Fanin

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.32	5.539	.452	30.675	238.73	150
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	45	45	348	5378	0
Mode:					
1					

X_2 : Fanout

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.698	3.074	.271	9.447	113.935	129
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	16	16	348	2148	0
Mode:					
1					

X_3 : Fanout to Lib. Mods.

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
.907	.972	.086	.944	107.148	129
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	5	5	117	227	0
Mode:					
•					

X_4 : No. Statements

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
21.953	28.02	2.467	785.138	127.635	129
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	198	197	2832	162670	0
Mode:					
3					

X_5 : McCabe

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
8.326	13.098	1.153	171.565	157.326	129
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	117	116	1074	30902	0
Mode:					
1					

X_6 : NPATH

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.872E7	1.786E8	15728053.31	3.191E16	621.951	129
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	1671070000	1671069999	3705126509	4.191E18	0
Mode:					
1					

Chef

X_1 : FanIn

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.656	4.014	.266	16.111	151.104	227
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	36	36	603	5243	0
Mode:					
1					

X_2 : Fanout

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
3.174	3.808	.276	14.504	120	190
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	29	29	603	4655	0
Mode:					
1					

X_3 : Fanout to Lib. Mods.

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
.621	1.1	.08	1.21	177.129	190
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	7	7	118	302	0
Mode:					
0					

X_4 : No. Statements

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
16.705	20.716	1.503	429.172	124.012	190
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	129	129	3174	134136	0
Mode:					
3					

X_5 : McCabe

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
6.832	8.533	.619	72.818	124.91	190
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	58	57	1298	22630	0
Mode:					
1					

X_6 : NPATH

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
1128.958	6951.444	505.643	4.832E7	615.74	189
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	64512	64512	213373	9325533617	0
Mode:					
2					

Compress

X_1 : FanIn

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
1.526	1.268	.206	1.607	83.065	38
Minimum:	Maximum:	Range:	Sum:	Sum Squared:	# Missing:
0	7	7	58	148	0
Mode:					
1					

X_2 : Fanout

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
3.625	5.584	1.396	31.183	154.047	16
Minimum:	Maximum:	Range:	Sum:	Sum Squared:	# Missing:
0	23	23	58	678	0
Mode:					
1					

X_3 : Fanout to Lib. Mods.

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.438	3.633	.908	13.196	149.03	16
Minimum:	Maximum:	Range:	Sum:	Sum Squared:	# Missing:
0	14	14	39	293	0
Mode:					
1					

X_4 : No. Statements

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
26.688	39.919	9.98	1593.562	149.581	16
Minimum:	Maximum:	Range:	Sum:	Sum Squared:	# Missing:
1	161	160	427	35299	0
Mode:					
3					

X_5 : McCabe

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
9.562	15.011	3.753	225.329	156.977	16
Minimum:	Maximum:	Range:	Sum:	Sum Squared:	# Missing:
1	62	61	153	4843	0
Mode:					
3					

X_6 : NPATH

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
1.679E7	6.544E7	1.636E7	4.283E15	389.766	16
Minimum:	Maximum:	Range:	Sum:	Sum Squared:	# Missing:
1	262130000	262129999	268647091	6.875E16	0
Mode:					
1					

Flex

X_1 : Fanin

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.024	2.232	.2	4.983	110.281	124
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	13	13	251	1121	0
Mode:					
1					

X_2 : Fanout

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.615	3.543	.362	12.555	135.522	96
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	24	24	251	1849	0
Mode:					
1					

X_3 : Fanout to Lib. Mods.

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
.76	1.064	.109	1.131	139.885	96
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	4	4	73	163	0
Mode:					
0					

X_4 : No. Statements

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
25.948	70.566	7.202	4979.587	271.953	96
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	597	597	2491	537697	0
Mode:					
2					

X_5 : McCabe

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
9.562	21.923	2.238	480.628	229.262	96
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	151	150	918	54438	0
Mode:					
1					

X_6 : NPATH

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
825885.979	5420184.202	556099.166	2.938E13	656.287	95
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	48798300	48798300	78459168	2.826E15	0
Mode:					
1					

MicroEmacs

X_1 : Fanin

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.725	5.869	.278	34.451	215.408	447
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	85	85	1218	18684	0
Mode:					
1					

X_2 : Fanout

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
3.123	6.118	.31	37.424	195.882	390
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	98	98	1218	18362	0
Mode:					
1					

X_3 : Fanout to Lib. Mods.

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
.867	1.444	.073	2.085	166.61	390
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	8	8	338	1104	0
Mode:					
0					

X_4 : No. Statements

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
15.985	24.127	1.222	582.113	150.939	390
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	217	217	6234	326090	0
Mode:					
1					

X_5 : McCabe

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
6.356	9.133	.462	83.412	143.682	390
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	81	80	2479	48205	0
Mode:					
1					

X_6 : NPATH

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2875.476	24422.46	1238.268	5.965E8	849.336	389
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	292608	292608	1118560	2.346E11	0
Mode:					
1					

Prolog

X_1 : Fanin

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.031	1.633	.129	2.668	80.418	161
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	9	9	327	1091	0
Mode:					
1					

X_2 : Fanout

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.795	6.69	.618	44.751	239.352	117
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	69	69	327	6105	0
Mode:					
1					

X_3 : Fanout to Lib. Mods.

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
.692	1.545	.143	2.387	223.178	117
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	11	11	81	333	0
Mode:					
0					

X_4 : No. Statements

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
28.957	124.468	11.507	15492.352	429.834	117
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	1316	1315	3388	1895220	0
Mode:					
2					

X_5 : McCabe

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
10.803	41.739	3.859	1742.108	386.346	117
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	435	434	1264	215740	0
Mode:					
1					

X_6 : NPATH

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2258877.243	2.233E7	2082552.703	4.988E14	988.672	115
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	238879000	238878999	259770883	5.745E16	0
Mode:					
1					

Top

X_1 : Fanin

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
1.741	2.02	.194	4.082	116.061	108
Minimum:	Maximum:	Range:	Sum:	Sum Squared:	# Missing:
0	14	14	188	764	0
Mode:					
1					

X_2 : Fanout

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
3.357	7.702	1.029	59.325	229.429	56
Minimum:	Maximum:	Range:	Sum:	Sum Squared:	# Missing:
0	58	58	188	3894	0
Mode:					
2					

X_3 : Fanout to Lib. Mods.

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.214	3.893	.52	15.153	175.8	56
Minimum:	Maximum:	Range:	Sum:	Sum Squared:	# Missing:
0	28	28	124	1108	0
Mode:					
1					

X_4 : No. Statements

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
12.875	36.25	4.844	1314.075	281.555	56
Minimum:	Maximum:	Range:	Sum:	Sum Squared:	# Missing:
1	272	271	721	81557	0
Mode:					
6					

X_5 : McCabe

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
4.946	10.88	1.454	118.379	219.961	56
Minimum:	Maximum:	Range:	Sum:	Sum Squared:	# Missing:
1	81	80	277	7881	0
Mode:					
1					

X_6 : NPATH

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
485.636	2853.938	384.825	8144960.754	587.67	55
Minimum:	Maximum:	Range:	Sum:	Sum Squared:	# Missing:
1	20822	20821	26710	452799228	0
Mode:					
1					

VN

X_1 : Fanin

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
3.261	4.776	.36	22.811	146.445	176
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	32	32	574	5864	0
Mode:					
1					

X_2 : Fanout

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
4.705	5.209	.472	27.135	110.717	122
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	35	35	574	5984	0
Mode:					
2					

X_3 : Fanout to Lib. Mods.

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.279	2.474	.224	6.12	108.566	122
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	11	11	278	1374	0
Mode:					
0					

X_4 : No. Statements

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
19.672	31.592	2.86	998.057	160.593	122
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	294	294	2400	167978	0
Mode:					
3					

X_5 : McCabe

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
7.336	8.937	.809	79.878	121.829	122
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	76	75	895	16231	0
Mode:					
2					

X_6 : NPATH

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
56712.73	493486.907	44678.205	2.435E11	870.152	122
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	5313020	5313020	6918953	2.986E13	0
Mode:					
2					

WM

X_1 : Fanin

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.314	2.159	.183	4.663	93.308	140
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	14	14	324	1398	0
Mode:					
1					

X_2 : Fanout

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
4.563	5.567	.661	30.992	121.995	71
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	30	30	324	3648	0
Mode:					
0					

X_3 : Fanout to Lib. Mods.

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.408	2.744	.326	7.531	113.942	71
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	12	12	171	939	0
Mode:					
0					

X_4 : No. Statements

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
23.085	36.231	4.3	1312.707	156.951	71
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	234	233	1639	129725	0
Mode:					
5					

X_5 : McCabe

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
9.197	13.245	1.572	175.418	144.006	71
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	81	80	653	18285	0
Mode:					
1					

X_6 : NPATH

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
20199.972	115922.435	13757.462	1.344E10	573.874	71
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	870944	870943	1434198	9.696E11	0
Mode:					
1					

Xscheme

X_1 : Fanin

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
3.018	9.801	.437	96.054	324.749	502
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	131	130	1515	52695	0
Mode:					
1					

X_2 : Fanout

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.683	2.729	.126	7.449	101.72	467
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	24	24	1253	6833	0
Mode:					
1					

X_3 : Fanout to Lib. Modules

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
.143	.592	.027	.351	412.723	467
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	9	9	67	173	0
Mode:					
0					

X_4 : No. Statements

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
8.362	17.617	.815	310.356	210.681	467
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	249	249	3905	177279	0
Mode:					
1					

X_5 : McCabe

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
3.595	6.734	.312	45.349	187.305	467
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	104	103	1679	27169	0
Mode:					
1					

X_6 : NPATH

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
1356.929	13718.368	634.81	1.882E8	1010.986	467
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	230384	230384	633686	8.856E10	0
Mode:					
1					

Yap

X_1 : Fanin

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
2.354	2.614	.218	6.832	111.027	144
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	20	20	339	1775	0
Mode:					
1					

X_2 : Fanout

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
3.11	4.122	.395	16.988	132.524	109
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	33	33	339	2889	0
Mode:					
1					

X_3 : Fanout to Lib. Mods.

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
.743	1.518	.145	2.304	204.25	109
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	9	9	81	309	0
Mode:					
0					

X_4 : No. Statements

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
12.147	16.102	1.542	259.275	132.562	109
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	91	91	1324	44084	0
Mode:					
1					

X_5 : McCabe

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
5.312	6.334	.607	40.124	119.248	109
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
1	34	33	579	7409	0
Mode:					
1					

X_6 : NPATH

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
732519.248	7626630.246	730498.692	5.817E13	1041.151	109
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
0	79626200	79626200	79844598	6.340E15	0
Mode:					
1					

Appendix D4 - Metric Correlations for Each Program in Test Set.

This appendix contains the linear correlation matrices for sets of metric values for each program in the test set. Values shown are those for the correlation coefficient, R.

Bib 62 Functions

Correlation Matrix for Variables: X_1 ... X_6

	No. Stat...	McCabe	Fanin	Fanout	Fanout t...	NPATH
No. Stateme...	1					
McCabe	.968	1				
Fanin	-.079	-.075	1			
Fanout	.74	.623	-.064	1		
Fanout to Lib...	.687	.548	-.097	.893	1	
NPATH	.447	.311	-.046	.488	.675	1

Bison 129 Functions

Correlation Matrix for Variables: X_1 ... X_6

	No. Stat...	McCabe	Fanin	Fanout	Fanout t...	NPATH
No. Stateme...	1					
McCabe	.949	1				
Fanin	-.015	.025	1			
Fanout	.373	.262	.003	1		
Fanout to Lib...	.216	.165	.12	.419	1	
NPATH	.303	.237	-.016	.058	.179	1

Chef 190 Functions

Correlation Matrix for Variables: X_1 ... X_6

	No. Stat...	McCabe	Fanin	Fanout	Fanout t...	NPATH
No. Stateme...	1					
McCabe	.949	1				
Fanin	-.127	-.125	1			
Fanout	.679	.596	-.122	1		
Fanout to Lib...	.364	.317	-.096	.382	1	
NPATH	.308	.433	-.063	.147	.068	1

Compress 16 Functions

Correlation Matrix for Variables: X_1 ... X_6

	No. Stat...	McCabe	Fanin	Fanout	Fanout t...	NPATH
No. Stateme...	1					
McCabe	.99	1				
Fanin	-.441	-.486	1			
Fanout	.933	.956	-.453	1		
Fanout to Lib...	.796	.845	-.373	.942	1	
NPATH	.904	.936	-.486	.929	.847	1

Flex 95 Functions

Correlation Matrix for Variables: X_1 ... X_6

	No. Stat...	McCabe	Fanin	Fanout	Fanout t...	NPATH
No. Stateme...	1					
McCabe	.95	1				
Fanin	-.056	-.08	1			
Fanout	.509	.569	-.109	1		
Fanout to Lib...	.343	.374	-.042	.603	1	
NPATH	.302	.349	-.069	.736	.362	1

Prolog 115 Functions

Correlation Matrix for Variables: X_1 ... X_6

	No. Stat...	McCabe	Fanin	Fanout	Fanout t...	NPATH
No. Stateme...	1					
McCabe	.957	1				
Fanin	.038	.063	1			
Fanout	.68	.635	.02	1		
Fanout to Lib...	.149	.092	-.02	.634	1	
NPATH	.391	.364	-.063	.191	-.04	1

Top 55 Functions

Correlation Matrix for Variables: X_1 ... X_6

	No. Stat...	McCabe	Fanin	Fanout	Fanout t...	NPATH
No. Stateme...	1					
McCabe	.947	1				
Fanin	-.036	-.049	1			
Fanout	.583	.451	-.084	1		
Fanout to Lib...	.548	.433	-.09	.826	1	
NPATH	.237	.171	-.049	.162	.156	1

Uemacs (MicroEmacs) 389 Functions

Correlation Matrix for Variables: X_1 ... X_6

	No. Stat...	McCabe	Fanin	Fanout	Fanout t...	NPATH
No. Stateme...	1					
McCabe	.948	1				
Fanin	.039	.047	1			
Fanout	.455	.422	-.056	1		
Fanout to Lib...	.54	.461	-.027	.491	1	
NPATH	.302	.28	-.021	.129	.249	1

VN 122 Functions

Correlation Matrix for Variables: X_1 ... X_6

	No. Stat...	McCabe	Fanin	Fanout	Fanout t...	NPATH
No. Stateme...	1					
McCabe	.965	1				
Fanin	-.001	-.008	1			
Fanout	.786	.743	-.041	1		
Fanout to Lib...	.463	.463	.001	.756	1	
NPATH	.286	.289	.005	.219	.268	1

WM 71 Functions

Correlation Matrix for Variables: X_1 ... X_6

	No. Stat...	McCabe	Fanin	Fanout	Fanout t...	NPATH
No. Stateme...	1					
McCabe	.974	1				
Fanin	-.096	-.104	1			
Fanout	.538	.554	-.093	1		
Fanout to Lib...	.304	.298	.043	.729	1	
NPATH	.19	.245	-.124	.377	.256	1

Xscheme 467 Functions

Correlation Matrix for Variables: X_1 ... X_6

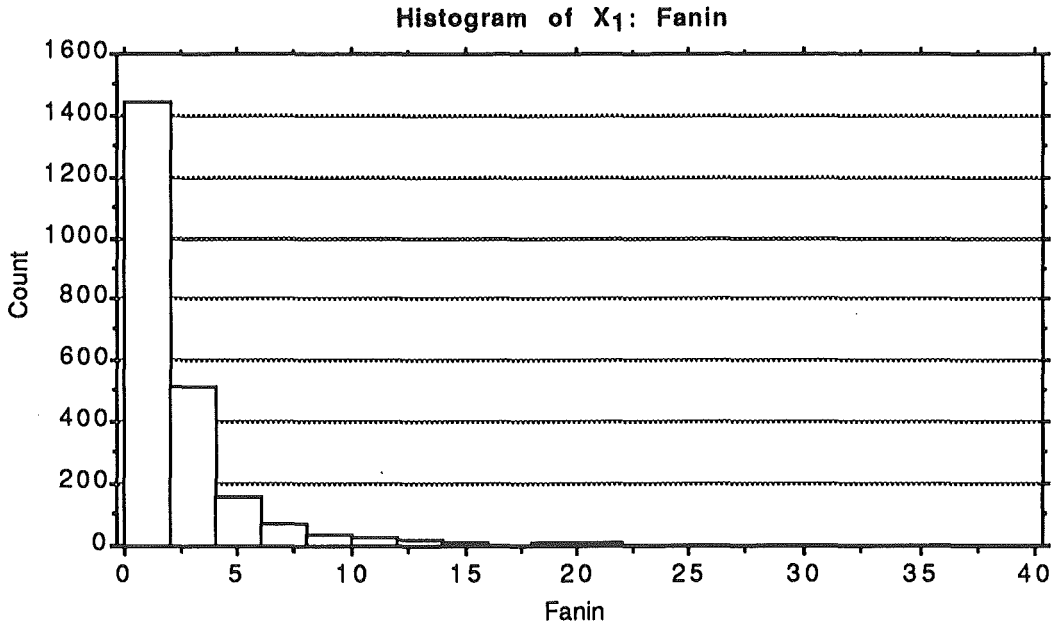
	No. Stat...	McCabe	Fanin	Fanout	Fanout t...	NPATH
No. Stateme...	1					
McCabe	.906	1				
Fanin	.004	.009	1			
Fanout	.666	.568	-.043	1		
Fanout to Lib...	.289	.242	.013	.377	1	
NPATH	.331	.406	.022	.108	.025	1

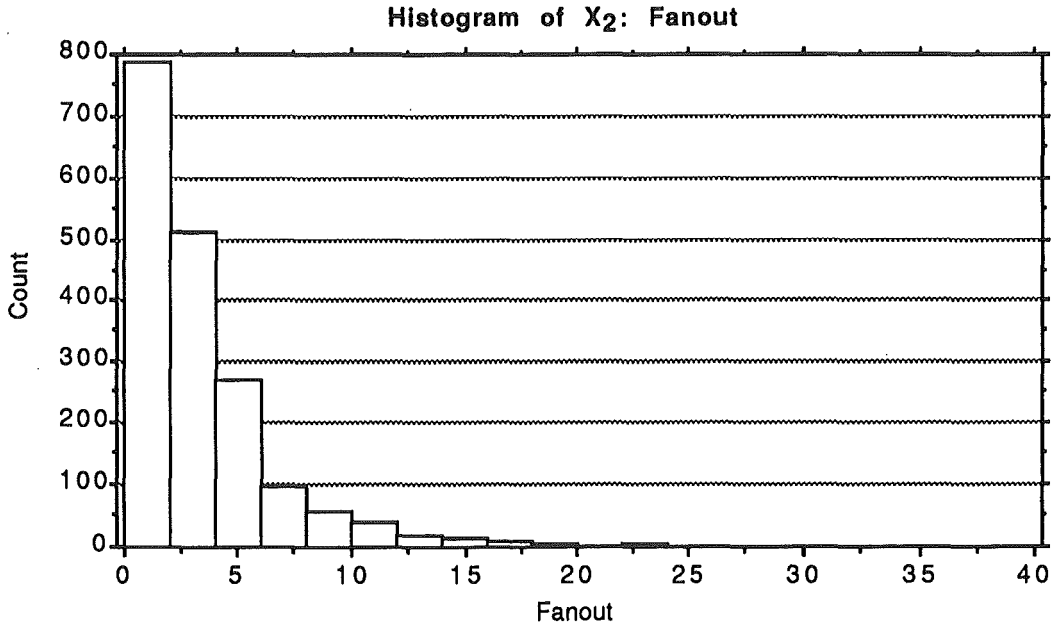
Correlation Matrix for Variables: X_1 ... X_6

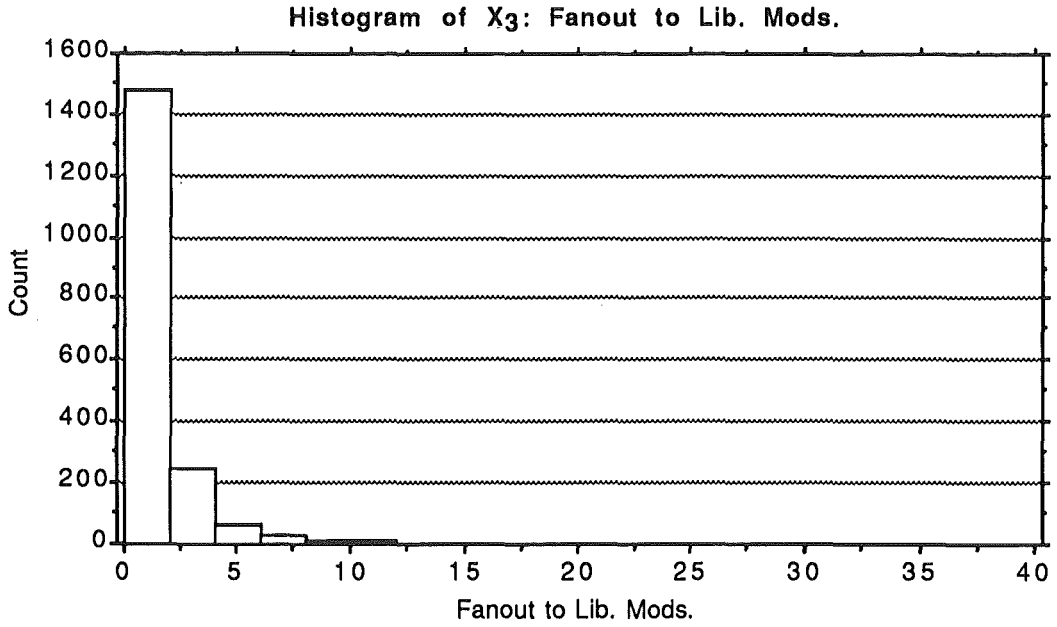
	No. Stat...	McCabe	Fanin	Fanout	Fanout t...	NPATH
No. Stateme...	1					
McCabe	.958	1				
Fanin	.019	.053	1			
Fanout	.479	.423	-.048	1		
Fanout to Lib...	.541	.475	-.063	.514	1	
NPATH	.474	.439	-.046	.279	.526	1

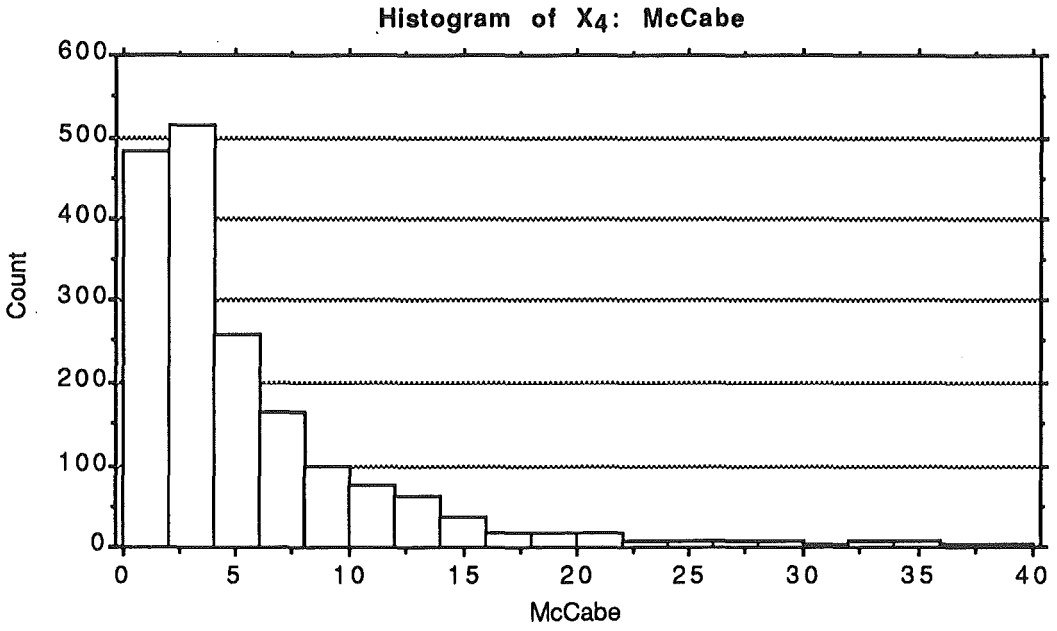
Appendix D5 - Frequency Distributions for Metric Values over all Programs.

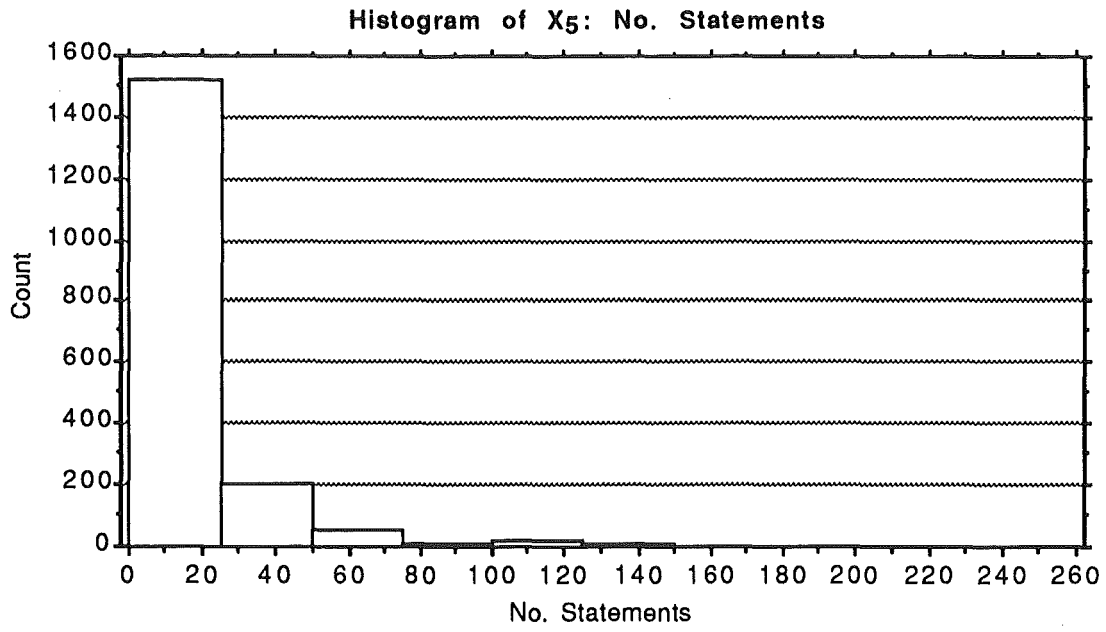
This appendix contains graphs of the frequencies of the metric values over the whole test set of 2310 modules for fanin, and 1825 modules for the other metrics.

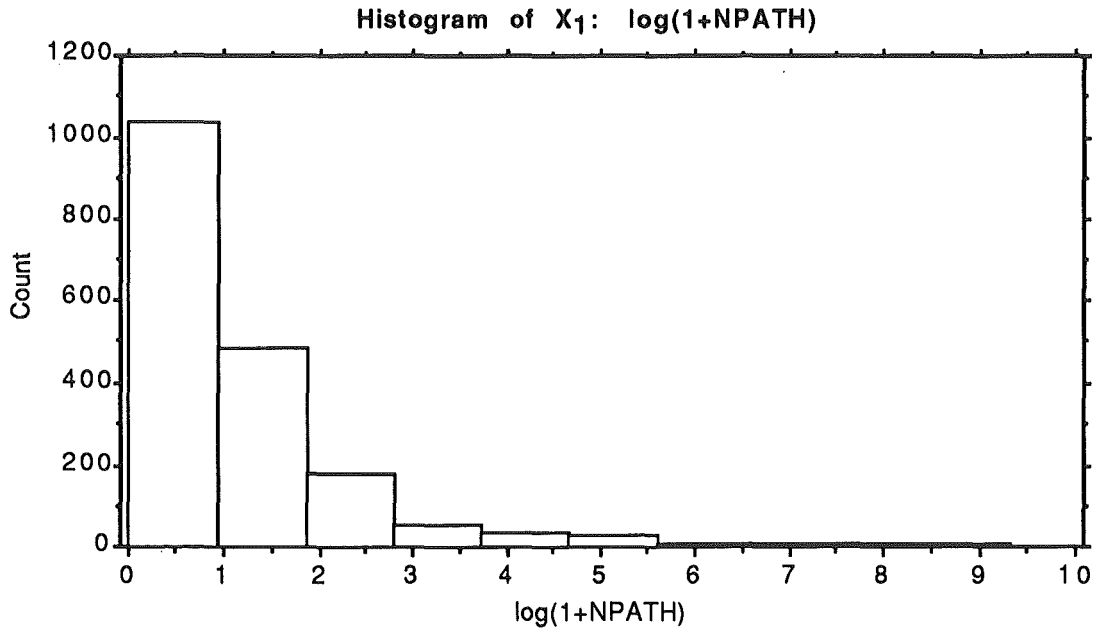





Histogram of X3: Fanout to Lib. Mods.
Count
1600
1400
1200
1000
800
600
400
200
0
0
5
10
15
20
25
30
35
40
Fanout to Lib. Mods.


Histogram of X4: McCabe
Count
600
500
400
300
200
100
0
0
5
10
15
20
25
30
35
40
McCabe


Histogram of X5: No. Statements
Count
1600
1400
1200
1000
800
600
400
200
0
0 20 40 60 80 100 120 140 160 180 200 220 240 260
No. Statements


Histogram of X1: log(1+NPATH)
Count
1200
1000
800
600
400
200
0
0 1 2 3 4 5 6 7 8 9 10
log(1+NPATH)

Appendix E. C Lexical Scanner and Parser.

This appendix includes copies of the basic C language lexical scanner and parser used for the McCabe's cyclomatic complexity, NPATH and number of statements metrics tools. The parser description does not include the modifications done to the "switch" statement for NPATH as described in Chapter 6, but instead uses the standard grammar for that statement decomposition. See Chapter 6 for details of the modification for NPATH.

The lexical scanner is present first, followed by the parser. All metric value calculating statements have been removed, but the modifications to the lexer and parser including typedef detection, and better constant recognition are included.

The C Language Lexical Scanner.

```
%{
/*
 * C Language Scanner
 * ==================
 *
 * Author: Tony Sanders.
 *
 * Modifications: Stephen Garner.
 * Date: Nov 1990
 *
 * Added cases for:   - float and exp types
 *                    - Added differentiation between identifers and typedefs
 *                      once a typedef name has been identified in the parser.
 *                    - Altered return values to be same as in parser.
 *
 * removed: const     { count(1); return CONST_TOKEN; }
 *
 */

#include <stdio.h>

#include "mc_c_parser.tab.h" /* for token values */

#define STRDUP(X) ((char *)strcpy(malloc(strlen(X)+1),X))

extern int yychar;
static int column = 0;
static int linenum = 1;
static char *the_string;
#define count(x) counter(x)
#define ERROR -1
#ifndef YYDEBUG
int     yydebug = 0;
#else
int     yydebug = 1;
#endif

/* Node structure for each node in the typedef list */

struct node
        {
        char entry[32];
        int depth;
        struct node *next;
        } *head_typedef_list = NULL;

int depth = 1;          /* current depth of statement nesting */

%}
alpha   [a-zA-Z]
digit   [0-9]
special [\_]
ident   (({alpha}|{special})({alpha}|{digit}|{special})*)
int     ({digit}+|{digit}+L|{digit}+l)
exp     ([Ee][-+]?{digit}+)
float   ({digit}+\.?{digit}*)
```

```
hex     (0x[0-9a-fA-F]+|0x[0-9a-fA-F]+L|0x[0-9a-fA-F]+l)
%p 3000
%x string
%%
^\#.*           { count(0); /* skip cpp lines */ }
[\ \n\t\v\f]+   { count(0); /* skip white space */ }
"/*"            { count(1); skipcomments(); }
">="            { count(1); return GREATER_THAN_EQUAL; }
"<="            { count(1); return LESS_THAN_EQUAL; }
"!="            { count(1); return NOT_EQUAL; }
"=="            { count(1); return EQUAL; }

"*="            { count(1); return MULTIPLY_ASSIGN; }
"/="            { count(1); return DIVIDE_ASSIGN; }
"%="            { count(1); return MODULUS_ASSIGN; }
"+="            { count(1); return ADD_ASSIGN; }
"-="            { count(1); return SUBTRACT_ASSIGN; }
"<<="           { count(1); return LEFT_SHIFT_ASSIGN; }
">>="           { count(1); return RIGHT_SHIFT_ASSIGN; }
"&="            { count(1); return AND_ASSIGN; }
"|="            { count(1); return OR_ASSIGN; }
"^="            { count(1); return XOR_ASSIGN; }

"<<"            { count(1); return LEFT_SHIFT; }
">>"            { count(1); return RIGHT_SHIFT; }
"++"            { count(1); return INCREMENT; }
"--"            { count(1); return DECREMENT; }
"->"            { count(1); return MEMBER_OF_STRUCT; }
"&&"            { count(1); return LOGICAL_AND; }
"||"            { count(1); return LOGICAL_OR; }


"("             { count(1); return LEFT_PARENTHESIS; }
","             { count(1); return COMMA;}
")"             { count(1); return RIGHT_PARENTHESIS; }
";"             { count(1); return SEMI_COLON; }
"{"             { count(1);
                  depth += 1;
                  return LEFT_BRACE; }
"}"             { count(1);
                  remove_obsolete_typedefs(depth);
                  depth -= 1;
                  return RIGHT_BRACE; }
"["             { count(1); return LEFT_BRACKET; }
"]"             { count(1); return RIGHT_BRACKET; }
"*"             { count(1); return STAR; }
"/"             { count(1); return DIVIDE; }
"+"             { count(1); return PLUS; }
"-"             { count(1); return MINUS; }
"%"             { count(1); return MODULUS; }
"|"             { count(1); return INCLUSIVE_OR;}
"^"             { count(1); return EXCLUSIVE_OR; }
"&"             { count(1); return AMPERSAND; }
"?"             { count(1); return CONDITION; }
":"             { count(1); return COLON; }
"!"             { count(1); return LOGICAL_NEGATE; }
"."             { count(1); return DOT; }
"~"             { count(1); return ONES_COMPLEMENT; }
"<"             { count(1); return LESS_THAN; }
">"             { count(1); return GREATER_THAN; }
"="             { count(1); return ASSIGN; }
"\""            { BEGIN(string);
                  count(1);
                  return STRING_LITERAL; }
<string>(\\\\)  { count(1); /* eat up all the \\ tokens */ }
<string>(\\\")  { count(1); /* eat up all the \" tokens */ }
<string>(\")    { count(1); BEGIN(0); /* got last " in string */ }
<string>.       { count(1);/* eat up all the other chars */ }

auto            { count(1); return AUTO_TOKEN; }
break           { count(1); return BREAK_TOKEN; }
case            { count(1); return CASE_TOKEN; }
char            { count(1); return CHAR_TOKEN; }
continue        { count(1); return CONTINUE_TOKEN; }
default         { count(1); return DEFAULT_TOKEN; }
do              { count(1); return DO_TOKEN; }
double          { count(1); return DOUBLE_TOKEN; }
else            { count(1); return ELSE_TOKEN; }
enum            { count(1); return ENUM_TOKEN; }
extern          { count(1); return EXTERN_TOKEN; }
float           { count(1); return FLOAT_TOKEN; }
for             { count(1); return FOR_TOKEN; }
goto            { count(1); return GOTO_TOKEN; }
if              { count(1); return IF_TOKEN; }
int             { count(1); return INT_TOKEN; }
long            { count(1); return LONG_TOKEN; }
register        { count(1); return REGISTER_TOKEN; }
```

```
return          { count(1); return RETURN_TOKEN; }
short           { count(1); return SHORT_TOKEN; }
sizeof          { count(1); return SIZEOF_TOKEN; }
static          { count(1); return STATIC_TOKEN; }
struct          { count(1); return STRUCT_TOKEN; }
switch          { count(1); return SWITCH_TOKEN; }
typedef         { count(1); return TYPEDEF_TOKEN; }
union           { count(1); return UNION_TOKEN; }
unsigned        { count(1); return UNSIGNED_TOKEN; }
void            { count(1); return VOID_TOKEN; }
volatile        { count(1); return VOLATILE_TOKEN; }
while           { count(1); return WHILE_TOKEN; }
PASCAL          { count(1); return PASCAL_TOKEN; }
FAR             { count(1); return FAR_TOKEN; }
NEAR            { count(1); return NEAR_TOKEN; }

"'\\''"{
        count(1);
        the_string = STRDUP(yytext+1);
        the_string[strlen(the_string)-1] = 0;
         yylval = (long)the_string;
        return CHARACTER_CONSTANT;
}

"'"[^\']*"'"   {
        count(1);
        the_string = STRDUP(yytext+1);
        the_string[strlen(the_string)-1] = 0;
         yylval = (long)the_string;
        return CHARACTER_CONSTANT;
}

{int}   {
        count(1);
        the_string = STRDUP(yytext+1);
        the_string[strlen(the_string)-1] = 0;
         yylval = (long)the_string;
        return INTEGER_CONSTANT;
}

{float} {
        count(1);
        the_string = STRDUP(yytext+1);
        the_string[strlen(the_string)-1] = 0;
         yylval = (long)the_string;
        return FLOATING_CONSTANT;
}
{hex}   {
        count(1);
        the_string = STRDUP(" 4294967296"); /* max long size + space for sign */
        sprintf(the_string,"%lx",yytext);
        yylval = (long)the_string;
        return HEX_CONSTANT;
}

{ident} {
        count(1);
        yylval = (long)STRDUP(yytext);

        /* check if yytext is in the list of known typedef names
           and if it is the return TYPEDEF_NAME, else IDENTIFIER */

        if (check_typedef_list(yytext))
                return TYPEDEF_NAME;
        else
                return IDENTIFIER;
}
.       { count(1); return ERROR; }
%%

/*
yywrap()
{
        return(1);
}
*/

/* Skip over comments. */
skipcomments()
{
    char c;

    while (1) {
       while ((c = input()) != '*')
            if (c == '\n') {
               column = 0;
               linenum++;
```

```
            }
            else if (c == '\t')
               column += 8 - (column % 8);
            else
               column++;
        if ((c = input()) == '/') {
            column++;
#ifdef LEXDEBUG
    printf ("symbol found: %s\n", "*/");
#endif
            return;
        }
        unput(c);
    }
}

/*ARGSUSED*/
counter (notwhite)
{
    register char *s;

#ifdef LEXDEBUG
    if (notwhite)
       printf ("## symbol found: %s\n", yytext);
#endif
    for (s = yytext; *s; s++)
       if (*s == '\n') {
            column = 0;
            linenum++;
       }
       else if (*s == '\t')
            column += 8 - (column % 8);
       else
            column++;
}

yyerror (s)
char   *s;
{
       fprintf (stdout, "YYERROR: %s: line %d col %d\n", s, linenum, column);
       fprintf (stdout, "YYERROR: yytext=`%s' symbol was (%d)\n", yytext, yychar);
}

/*****************************************************************************/

/* This function is called from the parser and adds a typedef name to the
   linked list containing know typedef names */

add_typedef_name(name)
char *name;
{
struct node *new, *current;

new = (struct node *) malloc (sizeof(struct node));
strcpy(new->entry,name);
new->depth = depth;
new->next = NULL;

current = head_typedef_list;

if (current == NULL)
  head_typedef_list = new;
else {
  while (current->next != NULL)
    current = current->next;

  current->next = new;
  }
}

/*****************************************************************************/

/* This function searches the typedef name list for the supplied name and
   returns 1 if it finds it, or 0 if it doesn't. */

int check_typedef_list(name)
char *name;
{
struct node *current;

for (current = head_typedef_list;current != NULL;current = current->next)
  if (!strcmp(name,current->entry))
    return(1);

return(0);
}
```

```
/*************************************************************************/

/* this function remove all the typedef names in the typedef list that
   occur at the nesting depth passed into the function */

remove_obsolete_typedefs()
{
struct node *back, *current, *node_to_free;

current = head_typedef_list;

while (current != NULL) {
  if (current->depth == depth) {
    if (current == head_typedef_list) {
      head_typedef_list = current->next;
      node_to_free = current;
      }
    else
      {
      back->next = current->next;
      node_to_free = current;
      }

    current = current->next;
    free(node_to_free);
    }
  else {
    back = current;
    current = current->next;
    }
  }
}
```

The C Language Parser.

```
%{

/* ****************************************************************

C Parser for YACC/Bison
=======================

Author: Satish Kumar

Notes:

- The grammar appears to be a faithful implementation of that found
  at the back of Kernighan and Ritchie, Second Edition.

Modifications: Stephen Garner (SRG). Date: November 1990

- Grammar changed to allow typedef declarations to be identified.
  This allows typedef names to be identified, and the scanner told
  about them so that next time it comes across that name it knows to
  return a TYPEDEF_NAME token to the parser and not an IDENTIFIER.

- Modification for the parsing of old style function declarations.

  **************************************************************** */

extern int add_typedef_name();

#define STRDUP(X) ((char *)strcpy(malloc(strlen(X)+1),X))

%}
%right    CONDITION COLON
%left     LOGICAL_OR
%left     LOGICAL_AND
%left     INCLUSIVE_OR
%left     EXCLUSIVE_OR
%left     AMPERSAND
%left     EQUAL NOT_EQUAL
%left     LEFT_SHIFT RIGHT_SHIFT
%left     PLUS MINUS
%left     STAR DIVIDE MODULUS
%left     INCREMENT DECREMENT
%left     DOT LEFT_PARENTHESIS LEFT_BRACKET
%right    LOGICAL_NEGATE ONES_COMPLEMENT
%left     LESS_THAN GREATER_THAN LESS_THAN_EQUAL GREATER_THAN_EQUAL
%left     COMMA
```

```
%token    SEMI_COLON
%token    LEFT_BRACE RIGHT_BRACE
%token    RIGHT_PARENTHESIS RIGHT_BRACKET
%right    ASSIGN MULTIPLY_ASSIGN DIVIDE_ASSIGN MODULUS_ASSIGN ADD_ASSIGN
%right    SUBTRACT_ASSIGN LEFT_SHIFT_ASSIGN RIGHT_SHIFT_ASSIGN
%right    AND_ASSIGN OR_ASSIGN XOR_ASSIGN
%token    MEMBER_OF_STRUCT
%token    AUTO_TOKEN                BREAK_TOKEN
%token    CASE_TOKEN                CHAR_TOKEN
%token    CHAR_TOKEN                CONST_TOKEN
%token    CONTINUE_TOKEN            DEFAULT_TOKEN
%token    DO_TOKEN                  DOUBLE_TOKEN
%token    ELSE_TOKEN                ENUM_TOKEN
%token    ENUM_TOKEN                EXTERN_TOKEN
%token    FLOAT_TOKEN               FOR_TOKEN
%token    GOTO_TOKEN                IF_TOKEN
%token    INT_TOKEN                 LONG_TOKEN
%token    REGISTER_TOKEN            RETURN_TOKEN
%token    SHORT_TOKEN               SIGNED_TOKEN
%token    SIZEOF_TOKEN              STATIC_TOKEN
%token    STRUCT_TOKEN              SWITCH_TOKEN
%token    TYPEDEF_TOKEN             UNION_TOKEN
%token    UNSIGNED_TOKEN            VOID_TOKEN
%token    VOLATILE_TOKEN            WHILE_TOKEN
%token    PASCAL_TOKEN              FAR_TOKEN            NEAR_TOKEN
%token    IDENTIFIER
%token    TYPEDEF_NAME
%token    INTEGER_CONSTANT          FLOATING_CONSTANT    HEX_CONSTANT
%token    CHARACTER_CONSTANT        ENUMERATION_CONSTANT
%token    STRING_LITERAL

%start    Translation_Unit

%%


Translation_Unit            : Translation_Unit
                              External_Declaration
                            | External_Declaration
                              ;

External_Declaration        : Function_Definition
                            | Declaration
                              ;

Function_Definition         : Declarator
                              Compound_Statement
                            | Declaration_Specifiers
                              Declarator
                              Declaration_List
                              Compound_Statement
                            | Declaration_Specifiers
                              Declarator
                              Compound_Statement
                            | Declarator
                              Declaration_List
                              Compound_Statement
                              ;

Declaration                 : Declaration_Specifiers
                              Init_Declarator_List
                              SEMI_COLON
                            | Declaration_Specifiers
                              SEMI_COLON
                              ;

Declaration_List            : Declaration
                            | Declaration_List
                              Declaration
                              ;

Declaration_Specifiers      : Storage_Class_Specifier
                            | Storage_Class_Specifier Declaration_Specifiers
                            | Type_Specifier
                            | Type_Specifier Declaration_Specifiers
                            | Type_Qualifier
                            | Type_Qualifier Declaration_Specifiers
                            | /* At this point we have a typedef declaration so
                                 we process it separately */
                              TYPEDEF_TOKEN

                              Type_Declaration_Specifiers
                              Type_Init_Declarator_List
                              ;

/* Theses are added for the decomposition of the typedef statement (SRG) */
```

```
Type_Declaration_Specifiers : Storage_Class_Specifier
                            | Storage_Class_Specifier Declaration_Specifiers
                            | Type_Specifier
                            | Type_Specifier Declaration_Specifiers
                            | Type_Qualifier
                            | Type_Qualifier Declaration_Specifiers
                            ;

Storage_Class_Specifier     : AUTO_TOKEN
                            | REGISTER_TOKEN
                            | STATIC_TOKEN
                            | EXTERN_TOKEN
/*                          | TYPEDEF_TOKEN */
                            /* Removed - typedef token has already been processed */
                            ;

Type_Specifier              : VOID_TOKEN
                            | CHAR_TOKEN
                            | SHORT_TOKEN
                            | INT_TOKEN
                            | LONG_TOKEN
                            | FLOAT_TOKEN
                            | DOUBLE_TOKEN
                            | SIGNED_TOKEN
                            | UNSIGNED_TOKEN
                            | PASCAL_TOKEN
                            | FAR_TOKEN
                            | NEAR_TOKEN
                            | Struct_Or_Union_Specifier
                            | Enum_Specifier
                            | TYPEDEF_NAME
                            ;

Type_Qualifier              : CONST_TOKEN
                            | VOLATILE_TOKEN
                            ;

Struct_Or_Union_Specifier   : Struct_Or_Union
                              IDENTIFIER
                            | Struct_Or_Union
                              TYPEDEF_NAME
                            | Struct_Or_Union
                              Identifier_Opt
                              LEFT_BRACE
                              Struct_Declaration_List
                              RIGHT_BRACE
                              ;

Identifier_Opt              : IDENTIFIER
                            |
                              ;

Struct_Or_Union             : STRUCT_TOKEN
                            | UNION_TOKEN
                              ;

Struct_Declaration_List     : Struct_Declaration_List
                              Struct_Declaration
                            | Struct_Declaration
                              ;

Init_Declarator_List        : Init_Declarator
                            | Init_Declarator_List
                              COMMA
                              Init_Declarator
                              ;

Init_Declarator             : Declarator
                            | Declarator
                              ASSIGN
                              Initializer
                              ;

/* added specifically for typedef declarators (SRG)*/

Type_Init_Declarator_List   : Type_Declarator
                            | Type_Init_Declarator_List
                              COMMA
                              Type_Declarator
                              ;

Struct_Declaration          : Specifier_Qualifier_List
                              Struct_Declarator_List
                              SEMI_COLON
                              ;

Specifier_Qualifier_List    : Type_Specifier
```

```
                              | Type_Specifier
                                Specifier_Qualifier_List
                              | Type_Qualifier
                              | Type_Qualifier
                                Specifier_Qualifier_List
                                ;

Struct_Declarator_List        : Struct_Declarator
                              | Struct_Declarator_List
                                COMMA
                                Struct_Declarator
                                ;

Struct_Declarator             : Declarator
                              | Declarator
                                COLON
                                Constant_Expression
                              | COLON
                                Constant_Expression
                                ;

Enum_Specifier                : ENUM_TOKEN
                                Identifier_Opt
                                LEFT_BRACE
                                Enumerator_List
                                RIGHT_BRACE
                              | ENUM_TOKEN
                                IDENTIFIER
                                ;

Enumerator_List               : Enumerator
                              | Enumerator_List
                                COMMA
                                Enumerator
                                ;

Enumerator                    : IDENTIFIER
                              | IDENTIFIER
                                ASSIGN
                                Constant_Expression
                                ;

Declarator                    : Pointer
                                Direct_Declarator
                              | Direct_Declarator
                                ;

Direct_Declarator             : IDENTIFIER
                              | LEFT_PARENTHESIS
                                Declarator
                                RIGHT_PARENTHESIS
                              | Direct_Declarator
                                LEFT_BRACKET
                                Constant_Expression_Opt
                                RIGHT_BRACKET
                              | Direct_Declarator
                                LEFT_PARENTHESIS
                                Parameter_Type_List
                                RIGHT_PARENTHESIS
                              | Direct_Declarator
                                LEFT_PARENTHESIS
                                Func_Identifier_List_Opt
                                RIGHT_PARENTHESIS
                                ;

/* added specifically for typedef declarators (SRG)*/

Type_Declarator               : Pointer
                                Type_Direct_Declarator
                              | Type_Direct_Declarator
                                ;

/* added specifically for typedef declarators (SRG)*/

Type_Direct_Declarator        : IDENTIFIER
                              | LEFT_PARENTHESIS
                                Type_Declarator
                                RIGHT_PARENTHESIS
                              | Type_Direct_Declarator
                                LEFT_BRACKET
                                Constant_Expression_Opt
                                RIGHT_BRACKET
                              | Type_Direct_Declarator
                                LEFT_PARENTHESIS
                                Identifier_List_Opt
                                RIGHT_PARENTHESIS
                                ;
```

```
Constant_Expression_Opt     : Constant_Expression
                            |
                              ;

Identifier_List_Opt         : Identifier_List
                            |
                              ;

Func_Identifier_List_Opt    : Func_Identifier_List
                            |
                              ;

Pointer                     : STAR
                            | STAR
                              Pointer
                            | STAR
                              Type_Qualifier_List
                            | STAR
                              Type_Qualifier_List
                              Pointer
                              ;

Type_Qualifier_List         : Type_Qualifier_List
                              Type_Qualifier
                            | Type_Qualifier
                              ;

Parameter_Type_List         : Parameter_List
                            | Parameter_List
                              COMMA
                              DOT
                              DOT
                              DOT
                              ;

Parameter_List              : Parameter_Declaration
                            | Parameter_List
                              COMMA
                              Parameter_Declaration
                              ;

Parameter_Declaration       : Declaration_Specifiers
                              Declarator
                            | Declaration_Specifiers
                              Abstract_Declarator
                            | Declaration_Specifiers
                              ;

Identifier_List             : IDENTIFIER
                            | Identifier_List
                              COMMA
                              IDENTIFIER
                              ;

Func_Identifier_List        : IDENTIFIER
                            | Func_Identifier_List
                              COMMA
                              IDENTIFIER
                              ;

Initializer                 : Assignment_Expression
                            | LEFT_BRACE
                              Initializer_List
                              RIGHT_BRACE
                            | LEFT_BRACE
                              Initializer_List
                              COMMA
                              RIGHT_BRACE
                              ;

Initializer_List            : Initializer
                            | Initializer_List
                              COMMA
                              Initializer
                              ;

Type_Name                   : Specifier_Qualifier_List
                              Abstract_Declarator
                            | Specifier_Qualifier_List
                              ;

Abstract_Declarator         : Pointer
                            | Pointer
                              Direct_Abstract_Declarator
                            | Direct_Abstract_Declarator
                              ;
```

```
Direct_Abstract_Declarator  : LEFT_PARENTHESIS
                              Abstract_Declarator
                              RIGHT_PARENTHESIS
                            | LEFT_PARENTHESIS
                              Parameter_Type_List_Opt
                              RIGHT_PARENTHESIS
                            | LEFT_BRACKET
                              Constant_Expression_Opt
                              RIGHT_BRACKET
                            | Direct_Abstract_Declarator
                              LEFT_BRACKET
                              Constant_Expression_Opt
                              RIGHT_BRACKET
                            | Direct_Abstract_Declarator
                              LEFT_PARENTHESIS
                              Parameter_Type_List_Opt
                              RIGHT_PARENTHESIS
                              ;

Parameter_Type_List_Opt     : Parameter_Type_List
                            |
                              ;

Statement                   : Labeled_Statement
                            | Expression_Statement
                            | Compound_Statement
                            | Selection_Statement
                            | Iteration_Statement
                            | Jump_Statement
                              ;

Labeled_Statement           : IDENTIFIER
                              COLON
                              Statement
                            | CASE_TOKEN
                              Constant_Expression
                              COLON
                              Statement
                            | DEFAULT_TOKEN
                              COLON
                              Statement
                              ;

Expression_Statement        : Expression_Opt
                              SEMI_COLON
                              ;

Expression_Opt              : Expression
                            |
                              ;

Compound_Statement          : LEFT_BRACE
                              RIGHT_BRACE
                            | LEFT_BRACE
                              Declaration_List
                              RIGHT_BRACE
                            | LEFT_BRACE
                              Statement_List
                              RIGHT_BRACE
                            | LEFT_BRACE
                              Declaration_List
                              Statement_List
                              RIGHT_BRACE
                              ;

Statement_List              : Statement_List Statement
                            | Statement
                              ;

Selection_Statement         : If_Statement
                            | If_Else_Statement
                            | Switch_Statement
                              ;

If_Statement                : IF_TOKEN
                              If_Condition
                              Statement
                              ;

If_Condition                : LEFT_PARENTHESIS
                              Expression
                              RIGHT_PARENTHESIS
                              ;

If_Else_Statement           : IF_TOKEN
                              If_Condition
```

```
                                IfStatement_Else
                                Statement
                                ;

IfStatement_Else            :   Statement
                                ELSE_TOKEN
                                ;

Switch_Statement            :   SWITCH_TOKEN
                                Switch_Expression
                                Statement
                                ;

Switch_Expression           :   LEFT_PARENTHESIS
                                Expression
                                RIGHT_PARENTHESIS
                                ;

Iteration_Statement         :   While_Statement
                            |   Do_While_Statement
                            |   For_Statement
                                ;

While_Statement             :   WHILE_TOKEN
                                While_Condition
                                Statement
                                ;

While_Condition             :   LEFT_PARENTHESIS
                                Expression
                                RIGHT_PARENTHESIS
                                ;

Do_While_Statement          :   DO_TOKEN
                                Do_Statement
                                Do_Condition
                                SEMI_COLON
                                ;

Do_Statement                :   Statement
                                WHILE_TOKEN
                                ;

Do_Condition                :   LEFT_PARENTHESIS
                                Expression
                                RIGHT_PARENTHESIS
                                ;

For_Statement               :   FOR_TOKEN
                                LEFT_PARENTHESIS
                                First_For_Expression
                                Second_For_Expression
                                Third_For_Expression
                                Statement
                                ;

First_For_Expression        :   Expression_Opt
                                SEMI_COLON
                                ;

Second_For_Expression       :   Expression_Opt
                                SEMI_COLON
                                ;

Third_For_Expression        :   Expression_Opt
                                RIGHT_PARENTHESIS
                                ;

Jump_Statement              :   GOTO_TOKEN
                                IDENTIFIER
                                SEMI_COLON
                            |   CONTINUE_TOKEN
                                SEMI_COLON
                            |   BREAK_TOKEN
                                SEMI_COLON
                            |   RETURN_TOKEN
                                Expression_Opt
                                SEMI_COLON
                                ;

Expression                  :   Assignment_Expression
                            |   Expression
                                COMMA
                                Assignment_Expression
                                ;

Assignment_Expression       :   Conditional_Expression
```

```
                        | Unary_Expression
                          Assignment_Operator
                          Assignment_Expression
                          ;

Assignment_Operator     : ASSIGN
                        | MULTIPLY_ASSIGN
                        | DIVIDE_ASSIGN
                        | MODULUS_ASSIGN
                        | ADD_ASSIGN
                        | SUBTRACT_ASSIGN
                        | LEFT_SHIFT_ASSIGN
                        | RIGHT_SHIFT_ASSIGN
                        | AND_ASSIGN
                        | OR_ASSIGN
                        | XOR_ASSIGN
                          ;

Conditional_Expression  : Logical_OR_Expression
                        | Logical_OR_Expression
                          CONDITION
                          Expression
                          COLON
                          Conditional_Expression
                          ;

Constant_Expression     : Conditional_Expression
                          ;

Logical_OR_Expression   : Logical_AND_Expression
                        | Logical_OR_Expression
                          LOGICAL_OR
                          Logical_AND_Expression
                          ;

Logical_AND_Expression  : Inclusive_OR_Expression
                        | Logical_AND_Expression
                          LOGICAL_AND
                          Inclusive_OR_Expression
                          ;

Inclusive_OR_Expression : Exclusive_OR_Expression
                        | Inclusive_OR_Expression
                          INCLUSIVE_OR
                          Exclusive_OR_Expression
                          ;

Exclusive_OR_Expression : AND_Expression
                        | Exclusive_OR_Expression
                          EXCLUSIVE_OR
                          AND_Expression
                          ;

AND_Expression          : Equality_Expression
                        | AND_Expression
                          AMPERSAND
                          Equality_Expression
                          ;

Equality_Expression     : Relational_Expression
                        | Equality_Expression
                          EQUAL
                          Relational_Expression
                        | Equality_Expression
                          NOT_EQUAL
                          Relational_Expression
                          ;

Relational_Expression   : Shift_Expression
                        | Relational_Expression
                          LESS_THAN
                          Shift_Expression
                        | Relational_Expression
                          GREATER_THAN
                          Shift_Expression
                        | Relational_Expression
                          LESS_THAN_EQUAL
                          Shift_Expression
                        | Relational_Expression
                          GREATER_THAN_EQUAL
                          Shift_Expression
                          ;

Shift_Expression        : Additive_Expression
                        | Shift_Expression
                          LEFT_SHIFT
                          Additive_Expression
```

```
                        | Shift_Expression
                          RIGHT_SHIFT
                          Additive_Expression
                          ;

Additive_Expression     : Multiplicative_Expression
                        | Additive_Expression
                          PLUS
                          Multiplicative_Expression
                        | Additive_Expression
                          MINUS
                          Multiplicative_Expression
                          ;

Multiplicative_Expression : Cast_Expression
                        | Multiplicative_Expression
                          STAR
                          Cast_Expression
                        | Multiplicative_Expression
                          DIVIDE
                          Cast_Expression
                        | Multiplicative_Expression
                          MODULUS
                          Cast_Expression
                          ;

Cast_Expression         : Unary_Expression
                        | LEFT_PARENTHESIS
                          Type_Name
                          RIGHT_PARENTHESIS
                          Cast_Expression
                          ;

Unary_Expression        : Postfix_Expression
                        | INCREMENT
                          Unary_Expression
                        | DECREMENT
                          Unary_Expression
                        | Unary_Operator
                          Cast_Expression
                        | SIZEOF_TOKEN
                          Unary_Expression
                        | SIZEOF_TOKEN
                          LEFT_PARENTHESIS
                          Type_Name
                          RIGHT_PARENTHESIS
                          ;

Unary_Operator          : AMPERSAND
                        | STAR
                        | PLUS
                        | MINUS
                        | LOGICAL_NEGATE
                        | ONES_COMPLEMENT
                          ;

Postfix_Expression      : Primary_Expression
                        | Postfix_Expression
                          LEFT_BRACKET
                          Expression
                          RIGHT_BRACKET
                        | Postfix_Expression
                          LEFT_PARENTHESIS
                          RIGHT_PARENTHESIS
                        | Postfix_Expression
                          LEFT_PARENTHESIS
                          Argument_Expression_List
                          RIGHT_PARENTHESIS
                        | Postfix_Expression
                          DOT
                          IDENTIFIER
                        | Postfix_Expression
                          MEMBER_OF_STRUCT
                          IDENTIFIER
                        | Postfix_Expression
                          MEMBER_OF_STRUCT
                          TYPEDEF_NAME
                        | Postfix_Expression
                          INCREMENT
                        | Postfix_Expression
                          DECREMENT
                          ;

Primary_Expression      : IDENTIFIER
                        | Constant
                        | STRING_LITERAL
                        | LEFT_PARENTHESIS
```

```
                                Expression
                                RIGHT_PARENTHESIS
                                ;

Argument_Expression_List    :   Assignment_Expression
                            |   Argument_Expression_List
                                COMMA
                                Assignment_Expression
                                ;

Constant                    :   INTEGER_CONSTANT
                            |   CHARACTER_CONSTANT
                            |   FLOATING_CONSTANT
                            |   ENUMERATION_CONSTANT
                            |   HEX_CONSTANT
                                ;


%%

main()
{
  yyparse();
  exit(0);
}
```

Appendix F. SMW Paper.

The following paper reproduced was presented at the 12th New Zealand Computer Conference held during the period 14-16 August,1991 in Dunedin, New Zealand.

The paper was published in the proceedings of that conference as:

Garner S ; Churcher N. A Software Metricians Workbench. Proc. 12th New Zealand Computer Conference, 1991, Wyvill G (editor), 1991

A Software Metrician's Workbench

Stephen Garner & Neville Churcher

Department of Computer Science, University of Canterbury,
Private Bag, Christchurch.

Abstract

We report progress to date on SMW—a software metrician's workbench. A set of tools for gathering, maintaining and reporting on various characteristics of software has been implemented. The data provided by SMW has applications in both the technical and managerial areas of software development. Some results for software written in C are presented.

1 Introduction

The image of the software development process has changed dramatically over the last two decades and the discipline of software engineering has emerged. An underlying principle is that software is a product which is the output of a sequence of well-defined production processes, enabling us to draw on the results and experiences of other branches of engineering. This picture encourages us to think of software in the same way that we view the products, such as bridges and automobiles, of other engineering processes. However, such a view is fundamentally flawed—software engineering *is* different in many ways.

An important aspect of any engineering discipline is the quantitative measurement of the relevant products and processes. One can readily measure properties such as the viscosity of a fluid, perform destruction testing on a scale model of a bridge, or compare the efficiencies of two different chemical reactions yielding some desired compound. Corresponding measurements of software and its development processes are much more difficult and quantitative results are hard to obtain. How can we tell if one program design is better than another, if one design methodology is more productive than another, if a programmer has done a good job of coding a particular design or if one programmer or program is better than another?

Attempts to answer questions such as these have led to the development of a number of software metrics, or measurable properties, and of models for their interpretation. The metrics used range from qualitative to quantitative and from simple to complex. For example, a simple metric for program source code would be the number of statements it contains while a more complex metric would take into account the nature and inter-relationships of the statements and data elements.

Software metrics are still in their infancy and ill-founded models and inappropriate application of metrics have caused the subject to be viewed justifiably with scepticism, if not derision, in the past.

However there is much to be gained from improved definitions, models and measurements of software quality. Two important principles that have emerged are that no single metric is sufficient to represent all the relevant properties of software and the software development process, and that properties of a given program should be compared against those of "similar" programs.

A successful software metrics program has a number of requirements. A large amount of data about the structure of program components and their relationships must be stored since these are necessary in order to compute metrics. Such data will change as the software evolves and a version control mechanism is necessary in order to address questions such as "did fixing that bug make the program simpler or more complex?"

In this paper we describe SMW, a system being developed at the University of Canterbury which will provide a flexible, extensible and powerful workbench for software metricians. The system is supported by a relational database and uses a variety of tools to generate and manipulate data. Some are written using standard Unix compiler generating tools, such as yacc and lex, and include tools which analyse directly program source code or design documents as well as others which analyse the output from Unix utilities such as cflow (a program flow graph generator) and cxref (a C program cross-referencer). The architecture and functions of SMW are shown in figure 1.

In the next section we briefly survey the available metrics and their applications while SMW itself is described in section 3. We are assembling a corpus of metric data which will ultimately be of use both in calibrating tools and in establishing baseline values for parameters in software process models. In section 4, some results are presented for a study of a number of C programs and indicate how SMW can provide support for product and process metrics at various stages of the software development cycle.

2 Software Metrics

A great deal has been written about software metrics and their applications [Côté 1988, Conte et al. 1986, Shepperd 1988]. The cost of correcting defects is much greater for those detected late in the development cycle. Design metrics [McCabe and Butler 1989, IST 1990] are available early in the development cycle, and are used to identify potential problem areas before coding begins and to allow quality assurance functions to be incorporated at an early stage. At the other end of the cycle, code metrics are used to plan structural testing activities and to identify potentially error-prone modules [Kafura & Reddy 1987]. Other applications include managerial functions such as cost, resource and size estimation and the ever-present bogey of programmer productivity.

It is generally believed that minimizing the complexity of software will lead to reduced costs over its life cycle. Typically, two-thirds of the cost of a software system is due to maintenance activities and the more complex the software, the longer it takes to comprehend so that changes can be made and the greater the probability of introducing further errors. Low complexity in

the earlier phases of the cycle leads to better control of entropy [Jensen and Tonies 1979]. Consequently, most models of the software development process involve measuring properties of software and asserting that they are good predictors of complexity.

Three broad classes of metric may be identified. The first is based on token counts. These include metrics such as the number of lines of code (LOC) or executable statements. Even with such apparently straightforward metrics a number of potential problems arise. For example, should declarations, comments and whitespace be counted—particularly in free-form languages such as C?

A family of token-based metrics which has become known as "software science" [Halstead 1977] is perhaps one of the most widely used. The number of distinct operators and operands and their frequencies of occurrence are used to measure the "length", "effort" and other properties of the software. Unfortunately, the psychological model of the way we understand programs, upon which software science is based, has been shown to be flawed [Lassez et al. 1981, Card and Agresti 1987] and the authors are unwilling to promote it. Nevertheless, software science has many enthusiastic supporters and publications continue to claim benefits from its application. It is our belief that these derive from the increased level discipline and commitment that accompany any formal quality control programme rather than from software science *per se*.

A second major class of metrics is based on the concept of program flow graphs. Nodes of the graph represent sequential code fragments and edges represent the branching control constructs. The archetypal metric in this class is cyclomatic complexity [McCabe 1976] which may be calculated in a number of equivalent ways. The NPATH metric [Nejmeh 1988] is typical of extensions which attempt to incorporate the effects of the extent to which structured programming is used.

The third class of metrics we wish to discuss are design metrics. These are typically derived from structure charts , design languages or recovered *post facto* from the code. These include the number of modules which call (fan-in) or are called by (fan-out) a given module and the extent to which the structure chart deviates from a pure tree structure [Ince and Hekmatpour 1988].

The complexity of a software system has two major components—the intra-modular complexity of each of its components and the inter-modular complexity of their interactions. Some metrics [Hemry and Kafura 1979, Card and Agresti 1988] attempt to combine these.

3 The SMW Architecture

The Software Metrician's Workbench supports the capture, maintenance and analysis of metric data. A central repository, implemented using an Ingres [Date 1987] relational database, contains data on current and previous revisions of programs. We have provided a number of tools for data capture, and the system is readily extensible. This is an important feature as application areas have differing requirements. A number of facilities for browsing the repository, reporting on particular programs and statistical analysis are also provided. Figure 1 shows the SMW architecture.

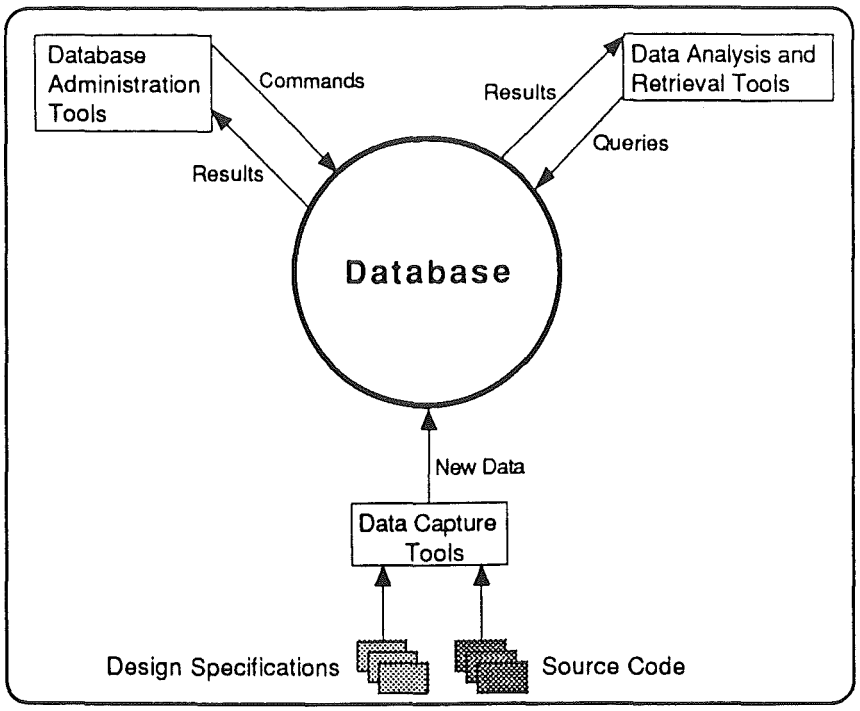


Figure 1. The SMW Architecture

The database stores structural information about modules, from which a program's call graph may be reconstructed, as well as values of individual metrics for modules which have been analyzed. The evolution of software may also be monitored as multiple versions of a programs and components may be maintained in the database.

We have implemented a range of data capture tools for programs written in C in order to show the generality of our approach. Structural information is readily obtained by parsing the output from standard Unix tools such as cflow, a program flow analyzer, or cxref, a cross-reference generator. Other metrics, such as the number of statements, cyclomatic complexity, and NPATH are obtained by appropriately instrumenting a C grammar and using the standard Unix compiler-construction tools yacc and lex.

One of the many difficulties associated with automated collection and processing of metric data is the validation and calibration of the tools used. One of our major objectives is to make available a reference set of metric values for widely-available programs. These have applications in teaching, establishing "typical" values for metrics and the comparison of metric analysis tools.

We currently provide tools to capture structural information for each module including fan-in & fan-out (both total and to library modules), level, frequency and parameters. Intra-module metrics currently supported are number of statements, cyclomatic complexity and NPATH.

The data may be accessed in a variety of ways. An interactive browser is available and one of its screens is shown in figure 2. The user selects a module and can see which other modules call, or are called by, it. A further option produces a screen showing details of available metric data for the selected module.

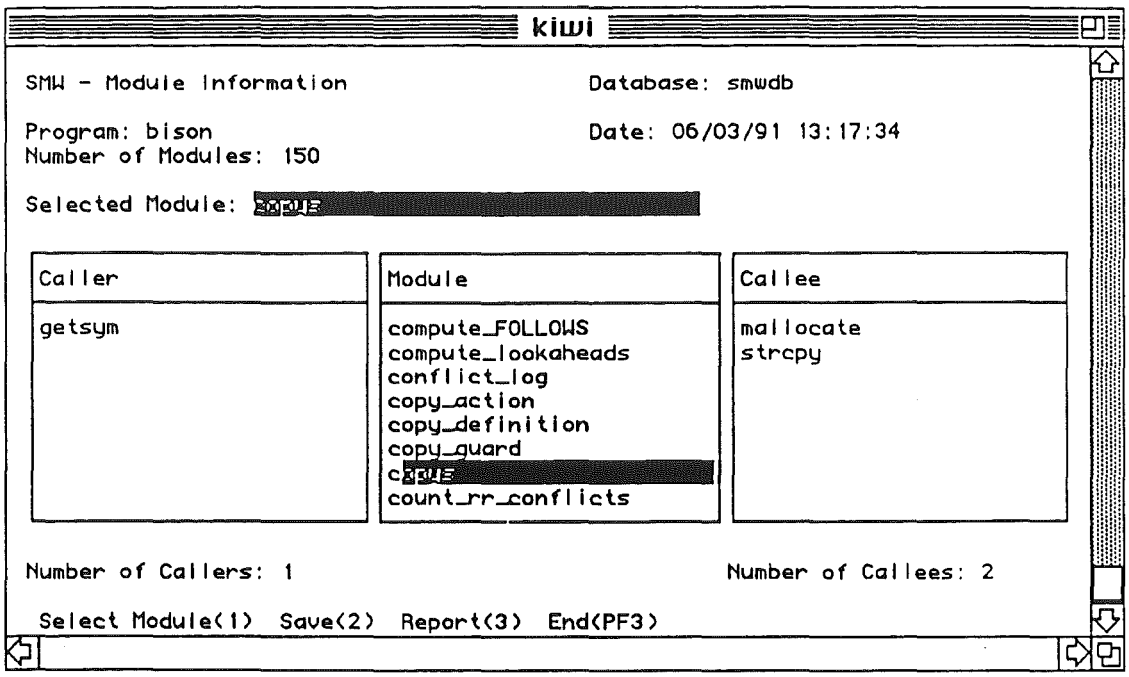


Figure 2. The SMW Browser

Common tasks such as the identification of potentially maintenance-prone modules whose metric values exceed some threshold value, typically $v > 10$ or NPATH > 100, are also supported by the browser. The user interface is written using the forms sub-system of Ingres, and is readily extensible. No modifications are required for many common extensions, such as the addition of a new metric tool.

A variety of reporting tools are also available and the Query-By-Forms, QUEL and SQL interfaces are available for *ad hoc* queries. Figures 3 to 5 were produced by exporting data to charting programs.

4 Results

In this section we present some representative results from our work to date. Space does not permit a complete analysis to be given—our aim is simply to display some of the capabilities of SMW. Figures 3 to 5 represent data from a sample of twelve widely available programs written in C. These include compiler-writing tools (flex and bison), editors (chef and microemacs), languages (xscheme and cprolog) and a newsreader (vn). Over 1700 distinct C functions are involved.

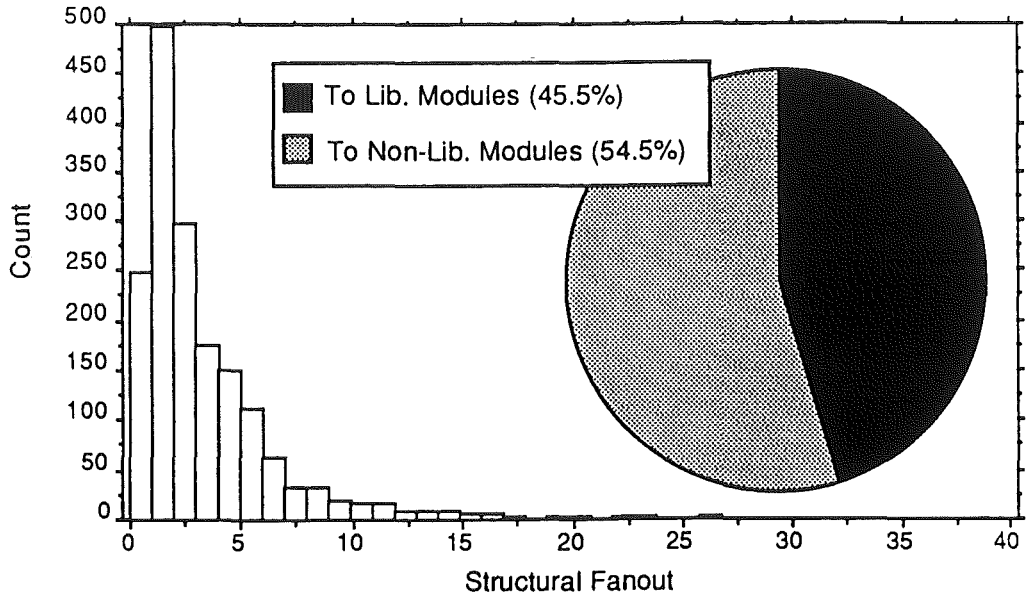


Figure 3. Distribution of Fan-Out

The structural fan-out (number of functions called) is shown in figure 3. The histogram shows the total fan-out while the inset pie-chart gives the breakdown into library (including system calls) and non-library modules. Functions with a fan-out greater that the "magic number" 7±2 [Miller 1957] are possibly over-complex. The corresponding results for fan-in show a similar distribution with most functions being called from only 1-3 other functions—indicating a tree-like structure. The authors were surprised by the number of functions never called at all!

Figures 4 and 5 show, respectively, the correlation of statement count and cyclomatic complexity, and statement count and fan-out. Typical features of such plots are visible—note that most functions have "small" values and that outliers are common. In fact the identification of outliers is one of the most useful applications of metrics analysis [IST 1990]. Further analysis may, of course, reveal that such functions are benign. Nevertheless, when finite resources are available for testing the outliers are a good place to begin.

5 Conclusion

We have constructed a flexible and extensible environment for the collection and analysis of software metric data. A variety of metrics have already been provided and it is straightforward to add new ones. A corpus of metric data is being established to act both as a reference set for the calibration of tools and as a research tool for the analysis of results to assist the development of improved metrics.

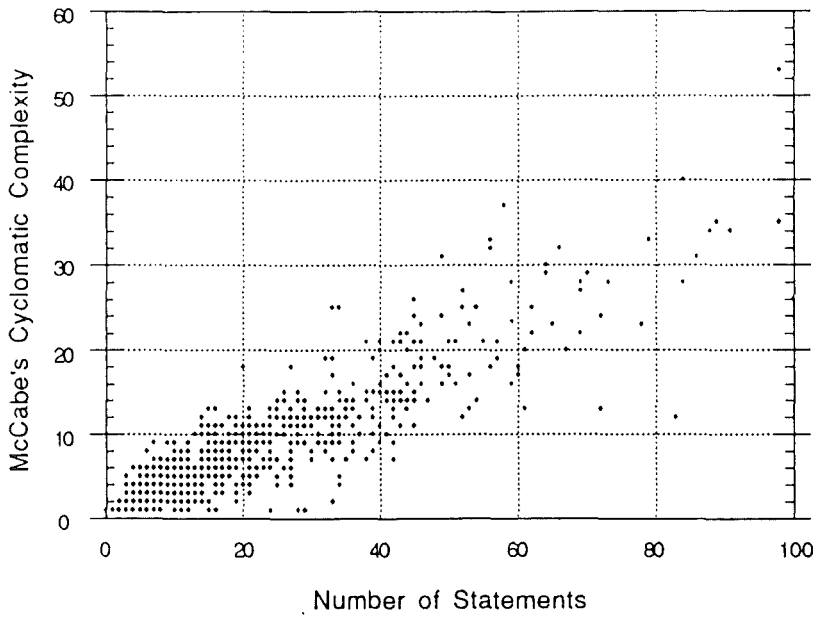


Figure 4. Correlation between Cyclomatic Complexity and Statement Count

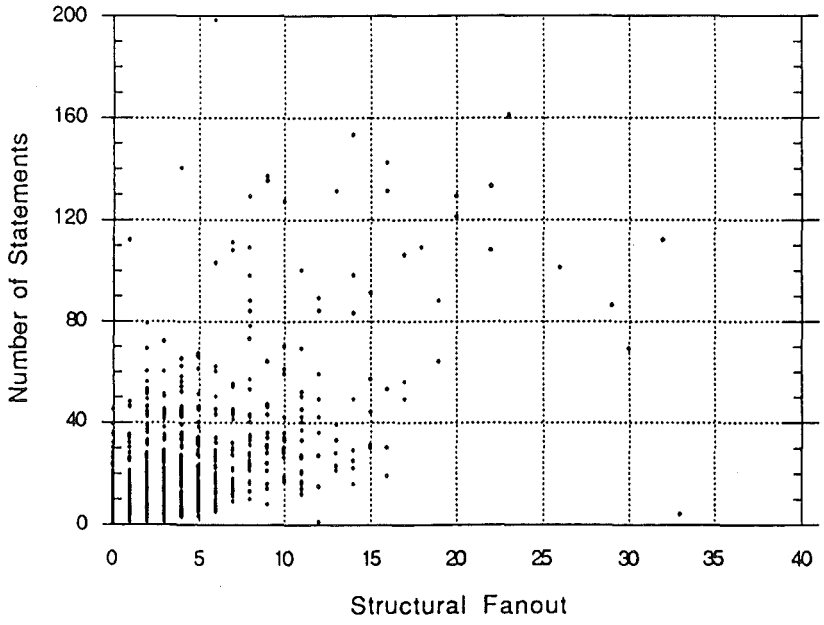


Figure 5. Correlation of Fan-Out and Statement Count

References

Card, D N and Agresti, W W 1987: Resolving the Software Science Anomaly. *J. Systems and Software*; Vol 7, 29-35

Card, D N and Agresti, W W 1988: Measuring Software Design Complexity. *J. Systems and Software*; Vol 8, 185-197

Conte, S D, Dunsmore, I and Shen H E 1986: *Software Engineering Metrics and Models*, Benjamin Cummings, Menlo Park

Côté, V, Bourque, P, Oligny, S, Rivard, N 1988: Software Metrics: An Overview of Recent Results. J. Systems and Software, Vol 8, 121-131

Date, C J 1987: *A guide to Ingres*, Addison-Wesley

Halstead, M H 1977: *Elements of Software Science*, Elsevier North-Holland, New York.

Henry, S and Kafura, D 1979: The Evaluation of Software Systems' Structure using Quantitative Software Metrics. *Software — Practice and Experience* Vol 14, No. 6, 561-573

Ince, D C and Hekmatpour S 1988: An Approach to Design based on Product Metrics, *Software Engineering Journal*, 53-56

IST 1990: Special issue on New Directions in Software Development, *Information & Software Technology*, Vol 32, No. 4.

Jensen, R and Tonies, C 1979: *Software Engineering*, Prentice-Hall, Englewood Cliffs NJ.

Kafura, D and Reddy, G 1987: The Use of Software Complexity Metrics in Software Maintenance, *IEEE Trans. Soft. Eng.*, Vol SE-13, No. 3, 335-343

Lassez, J-L, van der Knijff, Sheperd, J and Lassez, C 1981: A Critical Examination of Software Science, *J. Systems & Software*, Vol 2, 105-112

McCabe, T J 1976: A Complexity Measure. *IEEE Trans. Software Eng.*, Vol SE-2, No. 4, 308-320

McCabe, T J and Butler, C W 1989: Design Complexity and Testing, *Commun. ACM*, Vol. 32, No. 12, 1415-1425

Miller, G A 1957: The Magical Number 7 plus or minus two: Some Limits on our Capacity for Processing Information, *Psychological Review*, Vol. 63, 81-97

Nejmeh, Brian A 1988: NPATH: A Measure of Execution Path Complexity and its Applications. *Commun. ACM*, Vol 31, No.2, 188-200

Shepperd, M 1988: An Evaluation of Software Product Metrics. Information and Software Technology, Vol. 30, No. 3, 177-188

About the Authors

Stephen Garner has a B.Sc. in Computer Science from the University of Canterbury. This paper is partly based on work which he is currently completing towards an M.Sc. thesis. Neville Churcher holds B.Sc. (Hons) and Ph.D. degrees in physics from the University of Canterbury. After post-doctoral work in computational physics at the Cavendish Laboratory in Cambridge, he returned to New Zealand to take up a position with Adata Software. Since his appointment to the computer science department in 1987 his major areas of interest have been relational databases and software engineering.

Appendix G. Sample GNUPlot Scatter Plot.

The figure below is an example of the scatter plot able to be produced from using the SMW Browser interface's "Plot" option which produces a graphical plot of two metrics plotted against each other for a selected program version.

The sample plot contains data for the number of statements per modules plotted against that of the McCabe's cyclomatic complexity values for each module for the program "compress".